Arm® Architecture Registers

for A-profile architecture

arm

Copyright © 2018-2025 Arm Limited (or its affiliates). All rights reserved.
DDI 0601 (ID062025)

Arm® Architecture Registers
A-profile architecture

Copyright © 2018-2025 Arm Limited (or its affiliates). All rights reserved.

Release Information

For information on the change history and known issues for this release, see the **Release notes** in the **System Register XML for A-profile architecture (2025-06).**

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The validity, construction and performance of this License shall be governed by English Law.

The Arm corporate logo and words marked with ™ or © are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. You must follow the Arm's trademark usage guidelines
http://www.arm.com/company/policies/trademarks.

Copyright © 2010-2025 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-20349
8 March 2024

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Product Status

The information relating to the 2024 Extensions and the rest of the A-profile Architecture is at Beta quality. Beta quality means that all major features of the specification are described, but some details might be missing.

Web Address

`http://www.arm.com`

Progressive Terminology Commitment

Previous issues of this document included terms that can be offensive. We have replaced these terms. If you find offensive terms in this document, please contact `terms@arm.com`.

Feedback on this document

If you have any comments or queries about this document, create a ticket at `https://support.developer.arm.com`.

As part of the ticket, include:

- The title, *Arm® Architecture for A-profile architecture*.
- The number, DDI 0601.
- The section name to which your comments refer.
- The page number(s) to which your comments refer.
- The rule identifier(s) to which your comments refer if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Copyright © 2018-2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Copyright © 2018-2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

AArch64 Registers

ACCDATA_EL1: Accelerator Data

ACTLRMASK_EL1: Auxiliary Control Masking Register (EL1)

ACTLRMASK_EL2: Auxiliary Control Masking Register (EL2)

ACTLR_EL1: Auxiliary Control Register (EL1)

ACTLR_EL2: Auxiliary Control Register (EL2)

ACTLR_EL3: Auxiliary Control Register (EL3)

AFSR0_EL1: Auxiliary Fault Status Register 0 (EL1)

AFSR0_EL2: Auxiliary Fault Status Register 0 (EL2)

AFSR0_EL3: Auxiliary Fault Status Register 0 (EL3)

AFSR1_EL1: Auxiliary Fault Status Register 1 (EL1)

AFSR1_EL2: Auxiliary Fault Status Register 1 (EL2)

AFSR1_EL3: Auxiliary Fault Status Register 1 (EL3)

AIDR_EL1: Auxiliary ID Register

ALLINT: All Interrupt Mask Bit

AMAIR2_EL1: Extended Auxiliary Memory Attribute Indirection Register (EL1)

AMAIR2_EL2: Extended Auxiliary Memory Attribute Indirection Register (EL2)

AMAIR2_EL3: Extended Auxiliary Memory Attribute Indirection Register (EL3)

AMAIR_EL1: Auxiliary Memory Attribute Indirection Register (EL1)

AMAIR_EL2: Auxiliary Memory Attribute Indirection Register (EL2)

AMAIR_EL3: Auxiliary Memory Attribute Indirection Register (EL3)

AMCFGR_EL0: Activity Monitors Configuration Register

AMCG1IDR_EL0: Activity Monitors Counter Group 1 Identification Register

AMCGCR_EL0: Activity Monitors Counter Group Configuration Register

AMCNTENCLR0_EL0: Activity Monitors Count Enable Clear Register 0

AMCNTENCLR1_EL0: Activity Monitors Count Enable Clear Register 1

AMCNTENSET0_EL0: Activity Monitors Count Enable Set Register 0

AMCNTENSET1_EL0: Activity Monitors Count Enable Set Register 1

AMCR_EL0: Activity Monitors Control Register

AMEVCNTR0<n>_EL0: Activity Monitors Event Counter Registers 0

AMEVCNTR1<n>_EL0: Activity Monitors Event Counter Registers 1

AMEVCNTVOFF0<n>_EL2: Activity Monitors Event Counter Virtual Offset Registers 0

AMEVCNTVOFF1<n>_EL2: Activity Monitors Event Counter Virtual Offset Registers 1

AMEVTYPER0<n>_EL0: Activity Monitors Event Type Registers 0

AMEVTYPER1<n>_EL0: Activity Monitors Event Type Registers 1

AMUSERENR_EL0: Activity Monitors User Enable Register

APDAKeyHi_EL1: Pointer Authentication Key A for Data (bits[127:64])

APDAKeyLo_EL1: Pointer Authentication Key A for Data (bits[63:0])

APDBKeyHi_EL1: Pointer Authentication Key B for Data (bits[127:64])

APDBKeyLo_EL1: Pointer Authentication Key B for Data (bits[63:0])

APGAKeyHi_EL1: Pointer Authentication Key A for Code (bits[127:64])

APGAKeyLo_EL1: Pointer Authentication Key A for Code (bits[63:0])

APIAKeyHi_EL1: Pointer Authentication Key A for Instruction (bits[127:64])

APIAKeyLo_EL1: Pointer Authentication Key A for Instruction (bits[63:0])

APIBKeyHi_EL1: Pointer Authentication Key B for Instruction (bits[127:64])

APIBKeyLo_EL1: Pointer Authentication Key B for Instruction (bits[63:0])

BRBCR_EL1: Branch Record Buffer Control Register (EL1)

BRBCR_EL2: Branch Record Buffer Control Register (EL2)

BRBFCR_EL1: Branch Record Buffer Function Control Register

BRBIDR0_EL1: Branch Record Buffer ID0 Register

BRBINF<n>_EL1: Branch Record Buffer Information Register <n>

BRBINFINJ_EL1: Branch Record Buffer Information Injection Register

BRBSRC<n>_EL1: Branch Record Buffer Source Address Register <n>

BRBSRCINJ_EL1: Branch Record Buffer Source Address Injection Register

BRBTGT<n>_EL1: Branch Record Buffer Target Address Register <n>

BRBTGTINJ_EL1: Branch Record Buffer Target Address Injection Register

BRBTS_EL1: Branch Record Buffer Timestamp Register

CCSIDR2_EL1: Current Cache Size ID Register 2

CCSIDR_EL1: Current Cache Size ID Register

CLIDR_EL1: Cache Level ID Register

CNTFRQ_EL0: Counter-timer Frequency Register

CNTHCTL_EL2: Counter-timer Hypervisor Control Register

CNTHPS_CTL_EL2: Counter-timer Secure Physical Timer Control Register (EL2)

CNTHPS_CVAL_EL2: Counter-timer Secure Physical Timer CompareValue Register (EL2)

CNTHPS_TVAL_EL2: Counter-timer Secure Physical Timer TimerValue Register (EL2)

CNTHP_CTL_EL2: Counter-timer Hypervisor Physical Timer Control Register

CNTHP_CVAL_EL2: Counter-timer Physical Timer CompareValue Register (EL2)

CNTHP_TVAL_EL2: Counter-timer Physical Timer TimerValue Register (EL2)

CNTHVS_CTL_EL2: Counter-timer Secure Virtual Timer Control Register (EL2)

CNTHVS_CVAL_EL2: Counter-timer Secure Virtual Timer CompareValue Register (EL2)

CNTHVS_TVAL_EL2: Counter-timer Secure Virtual Timer TimerValue Register (EL2)

CNTHV_CTL_EL2: Counter-timer Virtual Timer Control Register (EL2)

CNTHV_CVAL_EL2: Counter-timer Virtual Timer CompareValue Register (EL2)

CNTHV_TVAL_EL2: Counter-timer Virtual Timer TimerValue Register (EL2)

CNTKCTL_EL1: Counter-timer Kernel Control Register

CNTPCTSS_EL0: Counter-timer Self-Synchronized Physical Count Register

CNTPCT_EL0: Counter-timer Physical Count Register

CNTPOFF_EL2: Counter-timer Physical Offset Register

CNTPS_CTL_EL1: Counter-timer Physical Secure Timer Control Register

CNTPS_CVAL_EL1: Counter-timer Physical Secure Timer CompareValue Register

CNTPS_TVAL_EL1: Counter-timer Physical Secure Timer TimerValue Register

CNTP_CTL_EL0: Counter-timer Physical Timer Control Register

CNTP_CVAL_EL0: Counter-timer Physical Timer CompareValue Register

CNTP_TVAL_EL0: Counter-timer Physical Timer TimerValue Register

CNTVCTSS_EL0: Counter-timer Self-Synchronized Virtual Count Register

CNTVCT_EL0: Counter-timer Virtual Count Register

CNTVOFF_EL2: Counter-timer Virtual Offset Register

CNTV_CTL_EL0: Counter-timer Virtual Timer Control Register

CNTV_CVAL_EL0: Counter-timer Virtual Timer CompareValue Register

CNTV_TVAL_EL0: Counter-timer Virtual Timer TimerValue Register

CONTEXTIDR_EL1: Context ID Register (EL1)

CONTEXTIDR_EL2: Context ID Register (EL2)

CPACRMASK_EL1: Architectural Feature Access Control Masking Register

CPACR_EL1: Architectural Feature Access Control Register

CPTRMASK_EL2: Architectural Feature Trap Masking Register

CPTR_EL2: Architectural Feature Trap Register (EL2)

CPTR_EL3: Architectural Feature Trap Register (EL3)

CSSELR_EL1: Cache Size Selection Register

CTR_EL0: Cache Type Register

CurrentEL: Current Exception Level

DACR32_EL2: Domain Access Control Register

DAIF: Interrupt Mask Bits

DBGAUTHSTATUS_EL1: Debug Authentication Status Register

DBGBCR<n>_EL1: Debug Breakpoint Control Registers

DBGBVR<n>_EL1: Debug Breakpoint Value Registers

DBGCLAIMCLR_EL1: Debug CLAIM Tag Clear Register

DBGCLAIMSET_EL1: Debug CLAIM Tag Set Register

DBGDTRRX_EL0: Debug Data Transfer Register, Receive

DBGDTRTX_EL0: Debug Data Transfer Register, Transmit

DBGDTR_EL0: Debug Data Transfer Register, half-duplex

DBGPRCR_EL1: Debug Power Control Register

DBGVCR32_EL2: Debug Vector Catch Register

DBGWCR<n>_EL1: Debug Watchpoint Control Registers

DBGWVR<n>_EL1: Debug Watchpoint Value Registers

DCZID_EL0: Data Cache Zero ID Register

DISR_EL1: Deferred Interrupt Status Register

DIT: Data Independent Timing

DLR_EL0: Debug Link Register

DSPSR_EL0: Debug Saved Program Status Register

ELR_EL1: Exception Link Register (EL1)

ELR_EL2: Exception Link Register (EL2)

ELR_EL3: Exception Link Register (EL3)

ERRIDR_EL1: Error Record ID Register

ERRSELR_EL1: Error Record Select Register

ERXADDR_EL1: Selected Error Record Address Register

ERXCTLR_EL1: Selected Error Record Control Register

ERXFR_EL1: Selected Error Record Feature Register

ERXGSR_EL1: Selected Error Record Group Status Register

ERXMISC0_EL1: Selected Error Record Miscellaneous Register 0

ERXMISC1_EL1: Selected Error Record Miscellaneous Register 1

ERXMISC2_EL1: Selected Error Record Miscellaneous Register 2

ERXMISC3_EL1: Selected Error Record Miscellaneous Register 3

ERXPFGCDN_EL1: Selected Pseudo-fault Generation Countdown Register

ERXPFGCTL_EL1: Selected Pseudo-fault Generation Control Register

ERXPFGF_EL1: Selected Pseudo-fault Generation Feature Register

ERXSTATUS_EL1: Selected Error Record Primary Status Register

ESR_EL1: Exception Syndrome Register (EL1)

ESR_EL2: Exception Syndrome Register (EL2)

ESR_EL3: Exception Syndrome Register (EL3)

FAR_EL1: Fault Address Register (EL1)

FAR_EL2: Fault Address Register (EL2)

FAR_EL3: Fault Address Register (EL3)

FGWTE3_EL3: Fine-Grained Write Traps EL3

FPCR: Floating-point Control Register

FPEXC32_EL2: Floating-Point Exception Control Register

FPMR: Floating-point Mode Register

FPSR: Floating-point Status Register

GCR_EL1: Tag Control Register.

GCSCRE0_EL1: Guarded Control Stack Control Register (EL0)

GCSCR_EL1: Guarded Control Stack Control Register (EL1)

GCSCR_EL2: Guarded Control Stack Control Register (EL2)

GCSCR_EL3: Guarded Control Stack Control Register (EL3)

GCSPR_EL0: Guarded Control Stack Pointer Register (EL0)

GCSPR_EL1: Guarded Control Stack Pointer Register (EL1)

GCSPR_EL2: Guarded Control Stack Pointer Register (EL2)

GCSPR_EL3: Guarded Control Stack Pointer Register (EL3)

GMID_EL1: Multiple tag transfer ID Register

GPCBW_EL3: Granule Protection Check Bypass Window Register (EL3)

GPCCR_EL3: Granule Protection Check Control Register (EL3)

GPTBR_EL3: Granule Protection Table Base Register

HACDBSBR_EL2: Hardware Accelerator for Cleaning Dirty State Base Register

HACDBSCONS_EL2: Hardware Accelerator for Cleaning Dirty State Consumer Register

HACR_EL2: Hypervisor Auxiliary Control Register

HAFGRTR_EL2: Hypervisor Activity Monitors Fine-Grained Read Trap Register

HCRX_EL2: Extended Hypervisor Configuration Register

HCR_EL2: Hypervisor Configuration Register

HDBSSBR_EL2: Hardware Dirty State Tracking Structure Base Register

HDBSSPROD_EL2: Hardware Dirty State Tracking Structure Producer Register

HDFGRTR2_EL2: Hypervisor Debug Fine-Grained Read Trap Register 2

HDFGRTR_EL2: Hypervisor Debug Fine-Grained Read Trap Register

HDFGWTR2_EL2: Hypervisor Debug Fine-Grained Write Trap Register 2

HDFGWTR_EL2: Hypervisor Debug Fine-Grained Write Trap Register

HFGITR2_EL2: Hypervisor Fine-Grained Instruction Trap Register 2

HFGITR_EL2: Hypervisor Fine-Grained Instruction Trap Register

HFGRTR2_EL2: Hypervisor Fine-Grained Read Trap Register 2

HFGRTR_EL2: Hypervisor Fine-Grained Read Trap Register

HFGWTR2_EL2: Hypervisor Fine-Grained Write Trap Register 2

HFGWTR_EL2: Hypervisor Fine-Grained Write Trap Register

HPFAR_EL2: Hypervisor IPA Fault Address Register

HSTR_EL2: Hypervisor System Trap Register

ICC_AP0R<n>_EL1: Interrupt Controller Active Priorities Group 0 Registers

ICC_AP1R<n>_EL1: Interrupt Controller Active Priorities Group 1 Registers

ICC_ASGI1R_EL1: Interrupt Controller Alias Software Generated Interrupt Group 1 Register

ICC_BPR0_EL1: Interrupt Controller Binary Point Register 0

ICC_BPR1_EL1: Interrupt Controller Binary Point Register 1

ICC_CTLR_EL1: Interrupt Controller Control Register (EL1)

ICC_CTLR_EL3: Interrupt Controller Control Register (EL3)

ICC_DIR_EL1: Interrupt Controller Deactivate Interrupt Register

ICC_EOIR0_EL1: Interrupt Controller End Of Interrupt Register 0

ICC_EOIR1_EL1: Interrupt Controller End Of Interrupt Register 1

ICC_HPPIR0_EL1: Interrupt Controller Highest Priority Pending Interrupt Register 0

ICC_HPPIR1_EL1: Interrupt Controller Highest Priority Pending Interrupt Register 1

ICC_IAR0_EL1: Interrupt Controller Interrupt Acknowledge Register 0

ICC_IAR1_EL1: Interrupt Controller Interrupt Acknowledge Register 1

ICC_IGRPEN0_EL1: Interrupt Controller Interrupt Group 0 Enable Register

ICC_IGRPEN1_EL1: Interrupt Controller Interrupt Group 1 Enable Register

ICC_IGRPEN1_EL3: Interrupt Controller Interrupt Group 1 Enable Register (EL3)

ICC_NMIAR1_EL1: Interrupt Controller Non-maskable Interrupt Acknowledge Register 1

ICC_PMR_EL1: Interrupt Controller Interrupt Priority Mask Register

ICC_RPR_EL1: Interrupt Controller Running Priority Register

ICC_SGI0R_EL1: Interrupt Controller Software Generated Interrupt Group 0 Register

ICC_SGI1R_EL1: Interrupt Controller Software Generated Interrupt Group 1 Register

ICC_SRE_EL1: Interrupt Controller System Register Enable Register (EL1)

ICC_SRE_EL2: Interrupt Controller System Register Enable Register (EL2)

ICC_SRE_EL3: Interrupt Controller System Register Enable Register (EL3)

ICH_AP0R<n>_EL2: Interrupt Controller Hyp Active Priorities Group 0 Registers

ICH_AP1R<n>_EL2: Interrupt Controller Hyp Active Priorities Group 1 Registers

ICH_EISR_EL2: Interrupt Controller End of Interrupt Status Register

ICH_ELRSR_EL2: Interrupt Controller Empty List Register Status Register

ICH_HCR_EL2: Interrupt Controller Hyp Control Register

ICH_LR<n>_EL2: Interrupt Controller List Registers

ICH_MISR_EL2: Interrupt Controller Maintenance Interrupt State Register

ICH_VMCR_EL2: Interrupt Controller Virtual Machine Control Register

ICH_VTR_EL2: Interrupt Controller VGIC Type Register

ICV_AP0R<n>_EL1: Interrupt Controller Virtual Active Priorities Group 0 Registers

ICV_AP1R<n>_EL1: Interrupt Controller Virtual Active Priorities Group 1 Registers

ICV_BPR0_EL1: Interrupt Controller Virtual Binary Point Register 0

ICV_BPR1_EL1: Interrupt Controller Virtual Binary Point Register 1

ICV_CTLR_EL1: Interrupt Controller Virtual Control Register

ICV_DIR_EL1: Interrupt Controller Deactivate Virtual Interrupt Register

ICV_EOIR0_EL1: Interrupt Controller Virtual End Of Interrupt Register 0

ICV_EOIR1_EL1: Interrupt Controller Virtual End Of Interrupt Register 1

ICV_HPPIR0_EL1: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

ICV_HPPIR1_EL1: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

ICV_IAR0_EL1: Interrupt Controller Virtual Interrupt Acknowledge Register 0

ICV_IAR1_EL1: Interrupt Controller Virtual Interrupt Acknowledge Register 1

ICV_IGRPEN0_EL1: Interrupt Controller Virtual Interrupt Group 0 Enable Register

ICV_IGRPEN1_EL1: Interrupt Controller Virtual Interrupt Group 1 Enable Register

ICV_NMIAR1_EL1: Interrupt Controller Virtual Non-maskable Interrupt Acknowledge Register 1

ICV_PMR_EL1: Interrupt Controller Virtual Interrupt Priority Mask Register

ICV_RPR_EL1: Interrupt Controller Virtual Running Priority Register

ID_AA64AFR0_EL1: AArch64 Auxiliary Feature Register 0

ID_AA64AFR1_EL1: AArch64 Auxiliary Feature Register 1

ID_AA64DFR0_EL1: AArch64 Debug Feature Register 0

ID_AA64DFR1_EL1: AArch64 Debug Feature Register 1

ID_AA64DFR2_EL1: AArch64 Debug Feature Register 2

ID_AA64FPFR0_EL1: AArch64 Floating-point Feature Register 0

ID_AA64ISAR0_EL1: AArch64 Instruction Set Attribute Register 0

ID_AA64ISAR1_EL1: AArch64 Instruction Set Attribute Register 1

ID_AA64ISAR2_EL1: AArch64 Instruction Set Attribute Register 2

ID_AA64ISAR3_EL1: AArch64 Instruction Set Attribute Register 3

ID_AA64MMFR0_EL1: AArch64 Memory Model Feature Register 0

ID_AA64MMFR1_EL1: AArch64 Memory Model Feature Register 1

ID_AA64MMFR2_EL1: AArch64 Memory Model Feature Register 2

ID_AA64MMFR3_EL1: AArch64 Memory Model Feature Register 3

ID_AA64MMFR4_EL1: AArch64 Memory Model Feature Register 4

ID_AA64PFR0_EL1: AArch64 Processor Feature Register 0

ID_AA64PFR1_EL1: AArch64 Processor Feature Register 1

ID_AA64PFR2_EL1: AArch64 Processor Feature Register 2

ID_AA64SMFR0_EL1: SME Feature ID Register 0

ID_AA64ZFR0_EL1: SVE Feature ID Register 0

ID_AFR0_EL1: AArch32 Auxiliary Feature Register 0

ID_DFR0_EL1: AArch32 Debug Feature Register 0

ID_DFR1_EL1: AArch32 Debug Feature Register 1

ID_ISAR0_EL1: AArch32 Instruction Set Attribute Register 0

ID_ISAR1_EL1: AArch32 Instruction Set Attribute Register 1

ID_ISAR2_EL1: AArch32 Instruction Set Attribute Register 2

ID_ISAR3_EL1: AArch32 Instruction Set Attribute Register 3

ID_ISAR4_EL1: AArch32 Instruction Set Attribute Register 4

ID_ISAR5_EL1: AArch32 Instruction Set Attribute Register 5

ID_ISAR6_EL1: AArch32 Instruction Set Attribute Register 6

ID_MMFR0_EL1: AArch32 Memory Model Feature Register 0

ID_MMFR1_EL1: AArch32 Memory Model Feature Register 1

ID_MMFR2_EL1: AArch32 Memory Model Feature Register 2

ID_MMFR3_EL1: AArch32 Memory Model Feature Register 3

ID_MMFR4_EL1: AArch32 Memory Model Feature Register 4

ID_MMFR5_EL1: AArch32 Memory Model Feature Register 5

ID_PFR0_EL1: AArch32 Processor Feature Register 0

ID_PFR1_EL1: AArch32 Processor Feature Register 1

ID_PFR2_EL1: AArch32 Processor Feature Register 2

IFSR32_EL2: Instruction Fault Status Register (EL2)

ISR_EL1: Interrupt Status Register

LORC_EL1: LORegion Control (EL1)

LOREA_EL1: LORegion End Address (EL1)

LORID_EL1: LORegionID (EL1)

LORN_EL1: LORegion Number (EL1)

LORSA_EL1: LORegion Start Address (EL1)

MAIR2_EL1: Extended Memory Attribute Indirection Register (EL1)

MAIR2_EL2: Extended Memory Attribute Indirection Register (EL2)

MAIR2_EL3: Extended Memory Attribute Indirection Register (EL3)

MAIR_EL1: Memory Attribute Indirection Register (EL1)

MAIR_EL2: Memory Attribute Indirection Register (EL2)

MAIR_EL3: Memory Attribute Indirection Register (EL3)

MDCCINT_EL1: Monitor DCC Interrupt Enable Register

MDCCSR_EL0: Monitor DCC Status Register

MDCR_EL2: Monitor Debug Configuration Register (EL2)

MDCR_EL3: Monitor Debug Configuration Register (EL3)

MDRAR_EL1: Monitor Debug ROM Address Register

MDSCR_EL1: Monitor Debug System Control Register

MDSELR_EL1: Breakpoint and Watchpoint Selection Register

MDSTEPOP_EL1: Monitor Debug Step Opcode Register

MECIDR_EL2: MEC Identification Register

MECID_A0_EL2: Alternate MECID for EL2 and EL2&0 translation regimes

MECID_A1_EL2: Alternate MECID for EL2&0 translation regimes.

MECID_P0_EL2: Primary MECID for EL2 and EL2&0 translation regimes

MECID_P1_EL2: Primary MECID for EL2&0 translation regimes

MECID_RL_A_EL3: Realm PA space Alternate MECID for EL3 stage 1 translation regime

MFAR_EL3: Physical Fault Address Register (EL3)

MIDR_EL1: Main ID Register

MPAM0_EL1: MPAM0 Register (EL1)

MPAM1_EL1: MPAM1 Register (EL1)

MPAM2_EL2: MPAM2 Register (EL2)

MPAM3_EL3: MPAM3 Register (EL3)

MPAMBW0_EL1: MPAM PE-side Maximum-bandwidth Control Register (EL0)

MPAMBW1_EL1: MPAM PE-side Maximum-bandwidth Control Register (EL1)

MPAMBW2_EL2: MPAM PE-side Maximum-bandwidth Control Register (EL2)

MPAMBW3_EL3: MPAM PE-side Maximum-bandwidth Control Register (EL3)

MPAMBWCAP_EL2: MPAM PE-side Maximum-bandwidth Limit Virtualization Register

MPAMBWIDR_EL1: MPAM PE-side Bandwidth Controls ID Register

MPAMBWSM_EL1: MPAM Streaming Mode Bandwidth Control Register (EL1)

MPAMHCR_EL2: MPAM Hypervisor Control Register (EL2)

MPAMIDR_EL1: MPAM ID Register (EL1)

MPAMSM_EL1: MPAM Streaming Mode Register

MPAMVPM0_EL2: MPAM Virtual PARTID Mapping Register 0

MPAMVPM1_EL2: MPAM Virtual PARTID Mapping Register 1

MPAMVPM2_EL2: MPAM Virtual PARTID Mapping Register 2

MPAMVPM3_EL2: MPAM Virtual PARTID Mapping Register 3

MPAMVPM4_EL2: MPAM Virtual PARTID Mapping Register 4

MPAMVPM5_EL2: MPAM Virtual PARTID Mapping Register 5

MPAMVPM6_EL2: MPAM Virtual PARTID Mapping Register 6

MPAMVPM7_EL2: MPAM Virtual PARTID Mapping Register 7

MPAMVPMV_EL2: MPAM Virtual Partition Mapping Valid Register

MPIDR_EL1: Multiprocessor Affinity Register

MVFR0_EL1: AArch32 Media and VFP Feature Register 0

MVFR1_EL1: AArch32 Media and VFP Feature Register 1

MVFR2_EL1: AArch32 Media and VFP Feature Register 2

NZCV: Condition Flags

OSDLR_EL1: OS Double Lock Register

OSDTRRX_EL1: OS Lock Data Transfer Register, Receive

OSDTRTX_EL1: OS Lock Data Transfer Register, Transmit

OSECCR_EL1: OS Lock Exception Catch Control Register

OSLAR_EL1: OS Lock Access Register

OSLSR_EL1: OS Lock Status Register

PAN: Privileged Access Never

PAR_EL1: Physical Address Register

PFAR_EL1: Physical Fault Address Register (EL1)

PFAR_EL2: Physical Fault Address Register (EL2)

PIRE0_EL1: Permission Indirection Register 0 (EL1)

PIRE0_EL2: Permission Indirection Register 0 (EL2)

PIR_EL1: Permission Indirection Register 1 (EL1)

PIR_EL2: Permission Indirection Register 2 (EL2)

PIR_EL3: Permission Indirection Register 3 (EL3)

PM: Profiling Exception Mask

PMBIDR_EL1: Profiling Buffer ID Register

PMBLIMITR_EL1: Profiling Buffer Limit Address Register

PMBMAR_EL1: Profiling Buffer Memory Attribute Register

PMBPTR_EL1: Profiling Buffer Write Pointer Register

PMBSR_EL1: Profiling Buffer Status/syndrome Register (EL1)

PMBSR_EL2: Profiling Buffer Syndrome Register (EL2)

PMBSR_EL3: Profiling Buffer Syndrome Register (EL3)

PMCCFILTR_EL0: Performance Monitors Cycle Count Filter Register

PMCCNTR_EL0: Performance Monitors Cycle Count Register

PMCCNTSVR_EL1: Performance Monitors Cycle Count Saved Value Register

PMCEID0_EL0: Performance Monitors Common Event Identification Register 0

PMCEID1_EL0: Performance Monitors Common Event Identification Register 1

PMCNTENCLR_EL0: Performance Monitors Count Enable Clear Register

PMCNTENSET_EL0: Performance Monitors Count Enable Set Register

PMCR_EL0: Performance Monitors Control Register

PMECR_EL1: Performance Monitors Extended Control Register (EL1)

PMEVCNTR<n>_EL0: Performance Monitors Event Count Registers

PMEVCNTSVR<n>_EL1: Performance Monitors Event Count Saved Value Registers

PMEVTYPER<n>_EL0: Performance Monitors Event Type Registers

PMIAR_EL1: Performance Monitors Instruction Address Register

PMICFILTR_EL0: Performance Monitors Instruction Counter Filter Register

PMICNTR_EL0: Performance Monitors Instruction Counter Register

PMICNTSVR_EL1: Performance Monitors Instruction Count Saved Value Register

PMINTENCLR_EL1: Performance Monitors Interrupt Enable Clear Register

PMINTENSET_EL1: Performance Monitors Interrupt Enable Set Register

PMMIR_EL1: Performance Monitors Machine Identification Register

PMOVSCLR_EL0: Performance Monitors Overflow Flag Status Clear Register

PMOVSSET_EL0: Performance Monitors Overflow Flag Status Set Register

PMSCR_EL1: Statistical Profiling Control Register (EL1)

PMSCR_EL2: Statistical Profiling Control Register (EL2)

PMSDSFR_EL1: Sampling Data Source Filter Register

PMSELR_EL0: Performance Monitors Event Counter Selection Register

PMSEVFR_EL1: Sampling Event Filter Register

PMSFCR_EL1: Sampling Filter Control Register

PMSICR_EL1: Sampling Interval Counter Register

PMSIDR_EL1: Sampling Profiling ID Register

PMSIRR_EL1: Sampling Interval Reload Register

PMSLATFR_EL1: Sampling Latency Filter Register

PMSNEVFR_EL1: Sampling Inverted Event Filter Register

PMSSCR_EL1: Performance Monitors Snapshot Status and Capture Register

PMSWINC_EL0: Performance Monitors Software Increment Register

PMUACR_EL1: Performance Monitors User Access Control Register

PMUSERENR_EL0: Performance Monitors User Enable Register

PMXEVCNTR_EL0: Performance Monitors Selected Event Count Register

PMXEVTYPER_EL0: Performance Monitors Selected Event Type Register

PMZR_EL0: Performance Monitors Zero with Mask

POR_EL0: Permission Overlay Register 0 (EL0)

POR_EL1: Permission Overlay Register 1 (EL1)

POR_EL2: Permission Overlay Register 2 (EL2)

POR_EL3: Permission Overlay Register 3 (EL3)

RCWMASK_EL1: Read Check Write Instruction Mask (EL1)

RCWSMASK_EL1: Software Read Check Write Instruction Mask (EL1)

REVIDR_EL1: Revision ID Register

RGSR_EL1: Random Allocation Tag Seed Register.

RMR_EL1: Reset Management Register (EL1)

RMR_EL2: Reset Management Register (EL2)

RMR_EL3: Reset Management Register (EL3)

RNDR: Random Number

RNDRRS: Random Number Full Entropy

RVBAR_EL1: Reset Vector Base Address Register (if EL2 and EL3 not implemented)

RVBAR_EL2: Reset Vector Base Address Register (if EL3 not implemented)

RVBAR_EL3: Reset Vector Base Address Register (if EL3 implemented)

S2PIR_EL2: Stage 2 Permission Indirection Register (EL2)

S2POR_EL1: Stage 2 Permission Overlay Register (EL1)

S3_<op1>_<Cn>_<Cm>_<op2>: IMPLEMENTATION DEFINED Registers

SCR_EL3: Secure Configuration Register

SCTLR2MASK_EL1: Extended System Control Masking Register (EL1)

SCTLR2MASK_EL2: Extended System Control Masking Register (EL2)

SCTLR2_EL1: System Control Register (EL1)

SCTLR2_EL2: System Control Register (EL2)

SCTLR2_EL3: System Control Register (EL3)

SCTLRMASK_EL1: System Control Masking Register (EL1)

SCTLRMASK_EL2: System Control Masking Register (EL2)

SCTLR_EL1: System Control Register (EL1)

SCTLR_EL2: System Control Register (EL2)

SCTLR_EL3: System Control Register (EL3)

SCXTNUM_EL0: EL0 Read/Write Software Context Number

SCXTNUM_EL1: EL1 Read/Write Software Context Number

SCXTNUM_EL2: EL2 Read/Write Software Context Number

SCXTNUM_EL3: EL3 Read/Write Software Context Number

SDER32_EL2: AArch32 Secure Debug Enable Register

SDER32_EL3: AArch32 Secure Debug Enable Register

SMCR_EL1: SME Control Register (EL1)

SMCR_EL2: SME Control Register (EL2)

SMCR_EL3: SME Control Register (EL3)

SMIDR_EL1: Streaming Mode Identification Register

SMPRIMAP_EL2: Streaming Mode Priority Mapping Register

SMPRI_EL1: Streaming Mode Priority Register

SPMACCESSR_EL1: System Performance Monitors Access Register (EL1)

SPMACCESSR_EL2: System Performance Monitors Access Register (EL2)

SPMACCESSR_EL3: System Performance Monitors Access Register (EL3)

SPMCFGR_EL1: System Performance Monitors Configuration Register

SPMCGCR<n>_EL1: System PMU Counter Group Configuration Registers

SPMCNTENCLR_EL0: System Performance Monitors Count Enable Clear Register

SPMCNTENSET_EL0: System Performance Monitors Count Enable Set Register

SPMCR_EL0: System Performance Monitor Control Register

SPMDEVAFF_EL1: System Performance Monitors Device Affinity Register

SPMDEVARCH_EL1: System Performance Monitors Device Architecture Register

SPMEVCNTR<n>_EL0: System Performance Monitors Event Count Register

SPMEVFILT2R<n>_EL0: System Performance Monitors Event Filter Control Register 2

SPMEVFILTR<n>_EL0: System Performance Monitors Event Filter Control Register

SPMEVTYPER<n>_EL0: System Performance Monitors Event Type Register

SPMIIDR_EL1: System PMU Implementation Identification Register

SPMINTENCLR_EL1: System Performance Monitors Interrupt Enable Clear Register

SPMINTENSET_EL1: System Performance Monitors Interrupt Enable Set Register

SPMOVSCLR_EL0: System Performance Monitors Overflow Flag Status Clear Register

SPMOVSSET_EL0: System Performance Monitors Overflow Flag Status Set Register

SPMROOTCR_EL3: System Performance Monitors Root and Realm Control Register

SPMSCR_EL1: System Performance Monitors Secure Control Register

SPMSELR_EL0: System Performance Monitors Select Register

SPMZR_EL0: System Performance Monitors Zero with Mask

SPSR_EL1: Saved Program Status Register (EL1)

SPSR_EL2: Saved Program Status Register (EL2)

SPSR_EL3: Saved Program Status Register (EL3)

SPSR_abt: Saved Program Status Register (Abort mode)

SPSR_fiq: Saved Program Status Register (FIQ mode)

SPSR_irq: Saved Program Status Register (IRQ mode)

SPSR_und: Saved Program Status Register (Undefined mode)

SPSel: Stack Pointer Select

SP_EL0: Stack Pointer (EL0)

SP_EL1: Stack Pointer (EL1)

SP_EL2: Stack Pointer (EL2)

SP_EL3: Stack Pointer (EL3)

SSBS: Speculative Store Bypass Safe

SVCR: Streaming Vector Control Register

TCO: Tag Check Override

TCR2MASK_EL1: Extended Translation Control Masking Register (EL1)

TCR2MASK_EL2: Extended Translation Control Masking Register (EL2)

TCR2_EL1: Extended Translation Control Register (EL1)

TCR2_EL2: Extended Translation Control Register (EL2)

TCRMASK_EL1: Translation Control Masking Register (EL1)

TCRMASK_EL2: Translation Control Masking Register (EL2)

TCR_EL1: Translation Control Register (EL1)

TCR_EL2: Translation Control Register (EL2)

TCR_EL3: Translation Control Register (EL3)

TFSRE0_EL1: Tag Fault Status Register (EL0).

TFSR_EL1: Tag Fault Status Register (EL1)

TFSR_EL2: Tag Fault Status Register (EL2)

TFSR_EL3: Tag Fault Status Register (EL3)

TPIDR2_EL0: EL0 Read/Write Software Thread ID Register 2

TPIDRRO_EL0: EL0 Read-Only Software Thread ID Register

TPIDR_EL0: EL0 Read/Write Software Thread ID Register

TPIDR_EL1: EL1 Software Thread ID Register

TPIDR_EL2: EL2 Software Thread ID Register

TPIDR_EL3: EL3 Software Thread ID Register

TRBBASER_EL1: Trace Buffer Base Address Register

TRBIDR_EL1: Trace Buffer ID Register

TRBLIMITR_EL1: Trace Buffer Limit Address Register

TRBMAR_EL1: Trace Buffer Memory Attribute Register

TRBMPAM_EL1: Trace Buffer MPAM Configuration Register

TRBPTR_EL1: Trace Buffer Write Pointer Register

TRBSR_EL1: Trace Buffer Status/syndrome Register (EL1)

TRBSR_EL2: Trace Buffer Syndrome Register (EL2)

TRBSR_EL3: Trace Buffer Syndrome Register (EL3)

TRBTRG_EL1: Trace Buffer Trigger Counter Register

TRCACATR<n>: Trace Address Comparator Access Type Register <n>

TRCACVR<n>: Trace Address Comparator Value Register <n>

TRCAUTHSTATUS: Trace Authentication Status Register

TRCAUXCTLR: Trace Auxiliary Control Register

TRCBBCTLR: Trace Branch Broadcast Control Register

TRCCCCTLR: Trace Cycle Count Control Register

TRCCIDCCTLR0: Trace Context Identifier Comparator Control Register 0

TRCCIDCCTLR1: Trace Context Identifier Comparator Control Register 1

TRCCIDCVR<n>: Trace Context Identifier Comparator Value Registers <n>

TRCCLAIMCLR: Trace Claim Tag Clear Register

TRCCLAIMSET: Trace Claim Tag Set Register

TRCCNTCTLR<n>: Trace Counter Control Register <n>

TRCCNTRLDVR<n>: Trace Counter Reload Value Register <n>

TRCCNTVR<n>: Trace Counter Value Register <n>

TRCCONFIGR: Trace Configuration Register

TRCDEVARCH: Trace Device Architecture Register

TRCDEVID: Trace Device Configuration Register

TRCEVENTCTL0R: Trace Event Control 0 Register

TRCEVENTCTL1R: Trace Event Control 1 Register

TRCEXTINSELR<n>: Trace External Input Select Register <n>

TRCIDR0: Trace ID Register 0

TRCIDR1: Trace ID Register 1

TRCIDR10: Trace ID Register 10

TRCIDR11: Trace ID Register 11

TRCIDR12: Trace ID Register 12

TRCIDR13: Trace ID Register 13

TRCIDR2: Trace ID Register 2

TRCIDR3: Trace ID Register 3

TRCIDR4: Trace ID Register 4

TRCIDR5: Trace ID Register 5

TRCIDR6: Trace ID Register 6

TRCIDR7: Trace ID Register 7

TRCIDR8: Trace ID Register 8

TRCIDR9: Trace ID Register 9

TRCIMSPEC0: Trace IMP DEF Register 0

TRCIMSPEC<n>: Trace IMP DEF Register <n>

TRCITECR_EL1: Instrumentation Trace Control Register (EL1)

TRCITECR_EL2: Instrumentation Trace Control Register (EL2)

TRCITEEDCR: Instrumentation Trace Extension External Debug Control Register

TRCOSLSR: Trace OS Lock Status Register

TRCPRGCTLR: Trace Programming Control Register

TRCQCTLR: Trace Q Element Control Register

TRCRSCTLR<n>: Trace Resource Selection Control Register <n>

TRCRSR: Trace Resources Status Register

TRCSEQEVR<n>: Trace Sequencer State Transition Control Register <n>

TRCSEQRSTEVR: Trace Sequencer Reset Control Register

TRCSEQSTR: Trace Sequencer State Register

TRCSSCCR<n>: Trace Single-shot Comparator Control Register <n>

TRCSSCSR<n>: Trace Single-shot Comparator Control Status Register <n>

TRCSSPCICR<n>: Trace Single-shot Processing Element Comparator Input Control Register <n>

TRCSTALLCTLR: Trace Stall Control Register

TRCSTATR: Trace Status Register

TRCSYNCPR: Trace Synchronization Period Register

TRCTRACEIDR: Trace ID Register

TRCTSCTLR: Trace Timestamp Control Register

TRCVICTLR: Trace ViewInst Main Control Register

TRCVIIECTLR: Trace ViewInst Include/Exclude Control Register

TRCVIPCSSCTLR: Trace ViewInst Start/Stop PE Comparator Control Register

TRCVISSCTLR: Trace ViewInst Start/Stop Control Register

TRCVMIDCCTLR0: Trace Virtual Context Identifier Comparator Control Register 0

TRCVMIDCCTLR1: Trace Virtual Context Identifier Comparator Control Register 1

TRCVMIDCVR<n>: Trace Virtual Context Identifier Comparator Value Register <n>

TRFCR_EL1: Trace Filter Control Register (EL1)

TRFCR_EL2: Trace Filter Control Register (EL2)

TTBR0_EL1: Translation Table Base Register 0 (EL1)

TTBR0_EL2: Translation Table Base Register 0 (EL2)

TTBR0_EL3: Translation Table Base Register 0 (EL3)

TTBR1_EL1: Translation Table Base Register 1 (EL1)

TTBR1_EL2: Translation Table Base Register 1 (EL2)

UAO: User Access Override

VBAR_EL1: Vector Base Address Register (EL1)

VBAR_EL2: Vector Base Address Register (EL2)

VBAR_EL3: Vector Base Address Register (EL3)

VDISR_EL2: Virtual Deferred Interrupt Status Register (EL2)

VDISR_EL3: Virtual Deferred Interrupt Status Register (EL3)

VMECID_A_EL2: Alternate MECID for EL1&0 stage 2 translation regime

VMECID_P_EL2: Primary MECID for EL1&0 stage 2 translation regime

VMPIDR_EL2: Virtualization Multiprocessor ID Register

VNCR_EL2: Virtual Nested Control Register

VPIDR_EL2: Virtualization Processor ID Register

VSESR_EL2: Virtual SError Exception Syndrome Register (EL2)

VSESR_EL3: Virtual SError Exception Syndrome Register (EL3)

VSTCR_EL2: Virtualization Secure Translation Control Register

VSTTBR_EL2: Virtualization Secure Translation Table Base Register

VTCR_EL2: Virtualization Translation Control Register

VTTBR_EL2: Virtualization Translation Table Base Register

ZCR_EL1: SVE Control Register (EL1)

ZCR_EL2: SVE Control Register (EL2)

ZCR_EL3: SVE Control Register (EL3)

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

2025-06_rel

AArch64 System Instructions

APAS: Associate PA space

AT S12E0R: Address Translate Stages 1 and 2 EL0 Read

AT S12E0W: Address Translate Stages 1 and 2 EL0 Write

AT S12E1R: Address Translate Stages 1 and 2 EL1 Read

AT S12E1W: Address Translate Stages 1 and 2 EL1 Write

AT S1E0R: Address Translate Stage 1 EL0 Read

AT S1E0W: Address Translate Stage 1 EL0 Write

AT S1E1A: Address Translate Stage 1 EL1 Without Permission checks

AT S1E1R: Address Translate Stage 1 EL1 Read

AT S1E1RP: Address Translate Stage 1 EL1 Read PAN

AT S1E1W: Address Translate Stage 1 EL1 Write

AT S1E1WP: Address Translate Stage 1 EL1 Write PAN

AT S1E2A: Address Translate Stage 1 EL2 Without Permission checks

AT S1E2R: Address Translate Stage 1 EL2 Read

AT S1E2W: Address Translate Stage 1 EL2 Write

AT S1E3A: Address Translate Stage 1 EL3 Without Permission checks

AT S1E3R: Address Translate Stage 1 EL3 Read

AT S1E3W: Address Translate Stage 1 EL3 Write

BRB IALL: Invalidate the Branch Record Buffer

BRB INJ: Branch Record Injection into the Branch Record Buffer

CFP RCTX: Control Flow Prediction Restriction by Context

COSP RCTX: Clear Other Speculative Prediction Restriction by Context

CPP RCTX: Cache Prefetch Prediction Restriction by Context

DC CGDSW: Clean of Data and Allocation Tags by Set/Way

DC CGDVAC: Clean of Data and Allocation Tags by VA to PoC

DC CGDVADP: Clean of Data and Allocation Tags by VA to PoDP

DC CGDVAOC: Clean of Data and Allocation Tags by VA to Outer Cache level

DC CGDVAP: Clean of Data and Allocation Tags by VA to PoP

DC CGSW: Clean of Allocation Tags by Set/Way

DC CGVAC: Clean of Allocation Tags by VA to PoC

DC CGVADP: Clean of Allocation Tags by VA to PoDP

DC CGVAP: Clean of Allocation Tags by VA to PoP

DC CIGDPAE: Clean and invalidate of data and allocation tags by PA to PoE

DC CIGDPAPA: Clean and Invalidate of Data and Allocation Tags by PA to PoPA

DC CIGDSW: Clean and Invalidate of Data and Allocation Tags by Set/Way

DC CIGDVAC: Clean and Invalidate of Data and Allocation Tags by VA to PoC

DC CIGDVAOC: Clean and Invalidate of Data and Allocation Tags by VA to Outer Cache level

DC CIGDVAPS: Clean and Invalidate of Data and Allocation Tags by VA to PoPS

DC CIGSW: Clean and Invalidate of Allocation Tags by Set/Way

DC CIGVAC: Clean and Invalidate of Allocation Tags by VA to PoC

DC CIPAE: Data or unified Cache line Clean and Invalidate by PA to PoE

DC CIPAPA: Data or unified Cache line Clean and Invalidate by PA to PoPA

DC CISW: Data or unified Cache line Clean and Invalidate by Set/Way

DC CIVAC: Data or unified Cache line Clean and Invalidate by VA to PoC

DC CIVAOC: Data or unified Cache line Clean and Invalidate by VA to Outer Cache level

DC CIVAPS: Clean and Invalidate of Data by VA to PoPS

DC CSW: Data or unified Cache line Clean by Set/Way

DC CVAC: Data or unified Cache line Clean by VA to PoC

DC CVADP: Data or unified Cache line Clean by VA to PoDP

DC CVAOC: Data or unified Cache line Clean by VA to Outer Cache level

DC CVAP: Data or unified Cache line Clean by VA to PoP

DC CVAU: Data or unified Cache line Clean by VA to PoU

DC GVA: Data Cache set Allocation Tag by VA

DC GZVA: Data Cache set Allocation Tags and Zero by VA

DC IGDSW: Invalidate of Data and Allocation Tags by Set/Way

DC IGDVAC: Invalidate of Data and Allocation Tags by VA to PoC

DC IGSW: Invalidate of Allocation Tags by Set/Way

DC IGVAC: Invalidate of Allocation Tags by VA to PoC

DC ISW: Data or unified Cache line Invalidate by Set/Way

DC IVAC: Data or unified Cache line Invalidate by VA to PoC

DC ZVA: Data Cache Zero by VA

DVP RCTX: Data Value Prediction Restriction by Context

GCSPOPCX: Guarded Control Stack Pop and Compare exception return record

GCSPOPM: Guarded Control Stack Pop

GCSPOPX: Guarded Control Stack Pop exception return record

GCSPUSHM: Guarded Control Stack Push

GCSPUSHX: Guarded Control Stack Push exception return record

GCSSS1: Guarded Control Stack Switch Stack 1

GCSSS2: Guarded Control Stack Switch Stack 2

IC IALLU: Instruction Cache Invalidate All to PoU

IC IALLUIS: Instruction Cache Invalidate All to PoU, Inner Shareable

IC IVAU: Instruction Cache line Invalidate by VA to PoU

SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2>, SYSP S1_<op1>_<Cn>_<Cm>_<op2>: IMPLEMENTATION DEFINED System instructions

TLBI ALLE1, TLBI ALLE1NXS: TLB Invalidate All, EL1

TLBI ALLE1IS, TLBI ALLE1ISNXS: TLB Invalidate All, EL1, Inner Shareable

TLBI ALLE1OS, TLBI ALLE1OSNXS: TLB Invalidate All, EL1, Outer Shareable

TLBI ALLE2, TLBI ALLE2NXS: TLB Invalidate All, EL2

TLBI ALLE2IS, TLBI ALLE2ISNXS: TLB Invalidate All, EL2, Inner Shareable

TLBI ALLE2OS, TLBI ALLE2OSNXS: TLB Invalidate All, EL2, Outer Shareable

TLBI ALLE3, TLBI ALLE3NXS: TLB Invalidate All, EL3

TLBI ALLE3IS, TLBI ALLE3ISNXS: TLB Invalidate All, EL3, Inner Shareable

TLBI ALLE3OS, TLBI ALLE3OSNXS: TLB Invalidate All, EL3, Outer Shareable

TLBI ASIDE1, TLBI ASIDE1NXS: TLB Invalidate by ASID, EL1

TLBI ASIDE1IS, TLBI ASIDE1ISNXS: TLB Invalidate by ASID, EL1, Inner Shareable

TLBI ASIDE1OS, TLBI ASIDE1OSNXS: TLB Invalidate by ASID, EL1, Outer Shareable

TLBI IPAS2E1, TLBI IPAS2E1NXS: TLB Invalidate by Intermediate Physical Address, Stage 2, EL1

TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS: TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS: TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

TLBI IPAS2LE1, TLBI IPAS2LE1NXS: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

TLBI PAALL: TLB Invalidate GPT Information by PA, All Entries, Local

TLBI PAALLOS: TLB Invalidate GPT Information by PA, All Entries, Outer Shareable

TLBI RIPAS2E1, TLBI RIPAS2E1NXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1

TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

TLBI RPALOS: TLB Range Invalidate GPT Information by PA, Last level, Outer Shareable

TLBI RPAOS: TLB Range Invalidate GPT Information by PA, Outer Shareable

TLBI RVAAE1, TLBI RVAAE1NXS: TLB Range Invalidate by VA, All ASID, EL1

TLBI RVAAE1IS, TLBI RVAAE1ISNXS: TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable

TLBI RVAAE1OS, TLBI RVAAE1OSNXS: TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

TLBI RVAALE1, TLBI RVAALE1NXS: TLB Range Invalidate by VA, All ASID, Last level, EL1

TLBI RVAALE1IS, TLBI RVAALE1ISNXS: TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

TLBI RVAALE1OS, TLBI RVAALE1OSNXS: TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

TLBI RVAE1, TLBI RVAE1NXS: TLB Range Invalidate by VA, EL1

TLBI RVAE1IS, TLBI RVAE1ISNXS: TLB Range Invalidate by VA, EL1, Inner Shareable

TLBI RVAE1OS, TLBI RVAE1OSNXS: TLB Range Invalidate by VA, EL1, Outer Shareable

TLBI RVAE2, TLBI RVAE2NXS: TLB Range Invalidate by VA, EL2

TLBI RVAE2IS, TLBI RVAE2ISNXS: TLB Range Invalidate by VA, EL2, Inner Shareable

TLBI RVAE2OS, TLBI RVAE2OSNXS: TLB Range Invalidate by VA, EL2, Outer Shareable

TLBI RVAE3, TLBI RVAE3NXS: TLB Range Invalidate by VA, EL3

TLBI RVAE3IS, TLBI RVAE3ISNXS: TLB Range Invalidate by VA, EL3, Inner Shareable

TLBI RVAE3OS, TLBI RVAE3OSNXS: TLB Range Invalidate by VA, EL3, Outer Shareable

TLBI RVALE1, TLBI RVALE1NXS: TLB Range Invalidate by VA, Last level, EL1

TLBI RVALE1IS, TLBI RVALE1ISNXS: TLB Range Invalidate by VA, Last level, EL1, Inner Shareable

TLBI RVALE1OS, TLBI RVALE1OSNXS: TLB Range Invalidate by VA, Last level, EL1, Outer Shareable

TLBI RVALE2, TLBI RVALE2NXS: TLB Range Invalidate by VA, Last level, EL2

TLBI RVALE2IS, TLBI RVALE2ISNXS: TLB Range Invalidate by VA, Last level, EL2, Inner Shareable

TLBI RVALE2OS, TLBI RVALE2OSNXS: TLB Range Invalidate by VA, Last level, EL2, Outer Shareable

TLBI RVALE3, TLBI RVALE3NXS: TLB Range Invalidate by VA, Last level, EL3

TLBI RVALE3IS, TLBI RVALE3ISNXS: TLB Range Invalidate by VA, Last level, EL3, Inner Shareable

TLBI RVALE3OS, TLBI RVALE3OSNXS: TLB Range Invalidate by VA, Last level, EL3, Outer Shareable

TLBI VAAE1, TLBI VAAE1NXS: TLB Invalidate by VA, All ASID, EL1

TLBI VAAE1IS, TLBI VAAE1ISNXS: TLB Invalidate by VA, All ASID, EL1, Inner Shareable

TLBI VAAE1OS, TLBI VAAE1OSNXS: TLB Invalidate by VA, All ASID, EL1, Outer Shareable

TLBI VAALE1, TLBI VAALE1NXS: TLB Invalidate by VA, All ASID, Last level, EL1

TLBI VAALE1IS, TLBI VAALE1ISNXS: TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

TLBI VAALE1OS, TLBI VAALE1OSNXS: TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

TLBI VAE1, TLBI VAE1NXS: TLB Invalidate by VA, EL1

TLBI VAE1IS, TLBI VAE1ISNXS: TLB Invalidate by VA, EL1, Inner Shareable

TLBI VAE1OS, TLBI VAE1OSNXS: TLB Invalidate by VA, EL1, Outer Shareable

TLBI VAE2, TLBI VAE2NXS: TLB Invalidate by VA, EL2

TLBI VAE2IS, TLBI VAE2ISNXS: TLB Invalidate by VA, EL2, Inner Shareable

TLBI VAE2OS, TLBI VAE2OSNXS: TLB Invalidate by VA, EL2, Outer Shareable

TLBI VAE3, TLBI VAE3NXS: TLB Invalidate by VA, EL3

TLBI VAE3IS, TLBI VAE3ISNXS: TLB Invalidate by VA, EL3, Inner Shareable

TLBI VAE3OS, TLBI VAE3OSNXS: TLB Invalidate by VA, EL3, Outer Shareable

TLBI VALE1, TLBI VALE1NXS: TLB Invalidate by VA, Last level, EL1

TLBI VALE1IS, TLBI VALE1ISNXS: TLB Invalidate by VA, Last level, EL1, Inner Shareable

TLBI VALE1OS, TLBI VALE1OSNXS: TLB Invalidate by VA, Last level, EL1, Outer Shareable

TLBI VALE2, TLBI VALE2NXS: TLB Invalidate by VA, Last level, EL2

TLBI VALE2IS, TLBI VALE2ISNXS: TLB Invalidate by VA, Last level, EL2, Inner Shareable

TLBI VALE2OS, TLBI VALE2OSNXS: TLB Invalidate by VA, Last level, EL2, Outer Shareable

TLBI VALE3, TLBI VALE3NXS: TLB Invalidate by VA, Last level, EL3

TLBI VALE3IS, TLBI VALE3ISNXS: TLB Invalidate by VA, Last level, EL3, Inner Shareable

TLBI VALE3OS, TLBI VALE3OSNXS: TLB Invalidate by VA, Last level, EL3, Outer Shareable

TLBI VMALLE1, TLBI VMALLE1NXS: TLB Invalidate by VMID, All at stage 1, EL1

TLBI VMALLE1IS, TLBI VMALLE1ISNXS: TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable

TLBI VMALLE1OS, TLBI VMALLE1OSNXS: TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable

TLBI VMALLS12E1, TLBI VMALLS12E1NXS: TLB Invalidate by VMID, All at Stage 1 and 2, EL1

TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS: TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable

TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS: TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable

TLBI VMALLWS2E1, TLBI VMALLWS2E1NXS: TLB Invalidate stage 2 dirty state by VMID, EL1&0

TLBI VMALLWS2E1IS, TLBI VMALLWS2E1ISNXS: TLB Invalidate stage 2 dirty state by VMID, EL1&0, Inner Shareable

TLBI VMALLWS2E1OS, TLBI VMALLWS2E1OSNXS: TLB Invalidate stage 2 write permission by VMID, EL1&0, Outer Shareable

TLBIP IPAS2E1, TLBIP IPAS2E1NXS: TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1

TLBIP IPAS2E1IS, TLBIP IPAS2E1ISNXS: TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

TLBIP IPAS2E1OS, TLBIP IPAS2E1OSNXS: TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

TLBIP IPAS2LE1, TLBIP IPAS2LE1NXS: TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1

TLBIP IPAS2LE1IS, TLBIP IPAS2LE1ISNXS: TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

TLBIP IPAS2LE1OS, TLBIP IPAS2LE1OSNXS: TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

TLBIP RIPAS2E1, TLBIP RIPAS2E1NXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1

TLBIP RIPAS2E1IS, TLBIP RIPAS2E1ISNXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

TLBIP RIPAS2E1OS, TLBIP RIPAS2E1OSNXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

TLBIP RIPAS2LE1, TLBIP RIPAS2LE1NXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

TLBIP RIPAS2LE1IS, TLBIP RIPAS2LE1ISNXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

TLBIP RIPAS2LE1OS, TLBIP RIPAS2LE1OSNXS: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

TLBIP RVAAE1, TLBIP RVAAE1NXS: TLB Range Invalidate by VA, All ASID, EL1

TLBIP RVAAE1IS, TLBIP RVAAE1ISNXS: TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable

TLBIP RVAAE1OS, TLBIP RVAAE1OSNXS: TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

TLBIP RVAALE1, TLBIP RVAALE1NXS: TLB Range Invalidate by VA, All ASID, Last level, EL1

TLBIP RVAALE1IS, TLBIP RVAALE1ISNXS: TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

TLBIP RVAALE1OS, TLBIP RVAALE1OSNXS: TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

TLBIP RVAE1, TLBIP RVAE1NXS: TLB Range Invalidate by VA, EL1

TLBIP RVAE1IS, TLBIP RVAE1ISNXS: TLB Range Invalidate by VA, EL1, Inner Shareable

TLBIP RVAE1OS, TLBIP RVAE1OSNXS: TLB Range Invalidate by VA, EL1, Outer Shareable

TLBIP RVAE2, TLBIP RVAE2NXS: TLB Range Invalidate by VA, EL2

TLBIP RVAE2IS, TLBIP RVAE2ISNXS: TLB Range Invalidate by VA, EL2, Inner Shareable

TLBIP RVAE2OS, TLBIP RVAE2OSNXS: TLB Range Invalidate by VA, EL2, Outer Shareable

TLBIP RVAE3, TLBIP RVAE3NXS: TLB Range Invalidate by VA, EL3

TLBIP RVAE3IS, TLBIP RVAE3ISNXS: TLB Range Invalidate by VA, EL3, Inner Shareable

TLBIP RVAE3OS, TLBIP RVAE3OSNXS: TLB Range Invalidate by VA, EL3, Outer Shareable

TLBIP RVALE1, TLBIP RVALE1NXS: TLB Range Invalidate by VA, Last level, EL1

TLBIP RVALE1IS, TLBIP RVALE1ISNXS: TLB Range Invalidate by VA, Last level, EL1, Inner Shareable

TLBIP RVALE1OS, TLBIP RVALE1OSNXS: TLB Range Invalidate by VA, Last level, EL1, Outer Shareable

TLBIP RVALE2, TLBIP RVALE2NXS: TLB Range Invalidate by VA, Last level, EL2

TLBIP RVALE2IS, TLBIP RVALE2ISNXS: TLB Range Invalidate by VA, Last level, EL2, Inner Shareable

TLBIP RVALE2OS, TLBIP RVALE2OSNXS: TLB Range Invalidate by VA, Last level, EL2, Outer Shareable

TLBIP RVALE3, TLBIP RVALE3NXS: TLB Range Invalidate by VA, Last level, EL3

TLBIP RVALE3IS, TLBIP RVALE3ISNXS: TLB Range Invalidate by VA, Last level, EL3, Inner Shareable

TLBIP RVALE3OS, TLBIP RVALE3OSNXS: TLB Range Invalidate by VA, Last level, EL3, Outer Shareable

TLBIP VAAE1, TLBIP VAAE1NXS: TLB Invalidate Pair by VA, All ASID, EL1

TLBIP VAAE1IS, TLBIP VAAE1ISNXS: TLB Invalidate Pair by VA, All ASID, EL1, Inner Shareable

TLBIP VAAE1OS, TLBIP VAAE1OSNXS: TLB Invalidate Pair by VA, All ASID, EL1, Outer Shareable

TLBIP VAALE1, TLBIP VAALE1NXS: TLB Invalidate Pair by VA, All ASID, Last level, EL1

TLBIP VAALE1IS, TLBIP VAALE1ISNXS: TLB Invalidate Pair by VA, All ASID, Last Level, EL1, Inner Shareable

TLBIP VAALE1OS, TLBIP VAALE1OSNXS: TLB Invalidate Pair by VA, All ASID, Last Level, EL1, Outer Shareable

TLBIP VAE1, TLBIP VAE1NXS: TLB Invalidate Pair by VA, EL1

TLBIP VAE1IS, TLBIP VAE1ISNXS: TLB Invalidate Pair by VA, EL1, Inner Shareable

TLBIP VAE1OS, TLBIP VAE1OSNXS: TLB Invalidate Pair by VA, EL1, Outer Shareable

TLBIP VAE2, TLBIP VAE2NXS: TLB Invalidate Pair by VA, EL2

TLBIP VAE2IS, TLBIP VAE2ISNXS: TLB Invalidate Pair by VA, EL2, Inner Shareable

TLBIP VAE2OS, TLBIP VAE2OSNXS: TLB Invalidate Pair by VA, EL2, Outer Shareable

TLBIP VAE3, TLBIP VAE3NXS: TLB Invalidate Pair by VA, EL3

TLBIP VAE3IS, TLBIP VAE3ISNXS: TLB Invalidate Pair by VA, EL3, Inner Shareable

TLBIP VAE3OS, TLBIP VAE3OSNXS: TLB Invalidate Pair by VA, EL3, Outer Shareable

TLBIP VALE1, TLBIP VALE1NXS: TLB Invalidate Pair by VA, Last level, EL1

TLBIP VALE1IS, TLBIP VALE1ISNXS: TLB Invalidate Pair by VA, Last level, EL1, Inner Shareable

TLBIP VALE1OS, TLBIP VALE1OSNXS: TLB Invalidate Pair by VA, Last level, EL1, Outer Shareable

TLBIP VALE2, TLBIP VALE2NXS: TLB Invalidate Pair by VA, Last level, EL2

TLBIP VALE2IS, TLBIP VALE2ISNXS: TLB Invalidate Pair by VA, Last level, EL2, Inner Shareable

TLBIP VALE2OS, TLBIP VALE2OSNXS: TLB Invalidate Pair by VA, Last level, EL2, Outer Shareable

TLBIP VALE3, TLBIP VALE3NXS: TLB Invalidate Pair by VA, Last level, EL3

TLBIP VALE3IS, TLBIP VALE3ISNXS: TLB Invalidate Pair by VA, Last level, EL3, Inner Shareable

TLBIP VALE3OS, TLBIP VALE3OSNXS: TLB Invalidate Pair by VA, Last level, EL3, Outer Shareable

TRCIT: Trace Instrumentation

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

2025-06_rel

ACCDATA_EL1, Accelerator Data

The ACCDATA_EL1 characteristics are:

Purpose

Holds the lower 32 bits of the data that is stored by an ST64BV0, Single-copy atomic 64-byte EL0 store instruction.

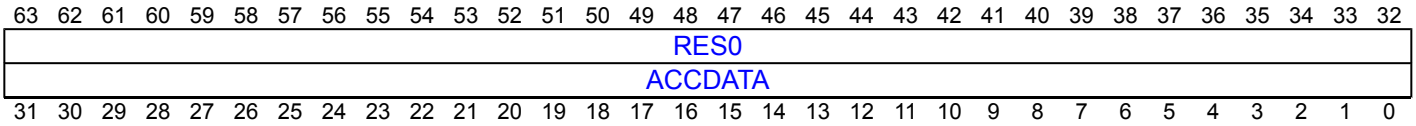
Configuration

This register is present only when FEAT_LS64_ACCDATA is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ACCDATA_EL1 are UNDEFINED.

Attributes

ACCDATA_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

ACCDATA, bits [31:0]

Accelerator Data field. Holds bits[31:0] of the data that is stored by an ST64BV0 instruction.

Accessing ACCDATA_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ACCDATA_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_LS64_ACCDATA) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ADEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nACCDATA_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ADEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ACCDATA_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ADEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ADEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ACCDATA_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACCDATA_EL1;
```

MSR ACCDATA_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_LS64_ACCDATA) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ADEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nACCDATA_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ADEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ACCDATA_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ADEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ADEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ACCDATA_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ACCDATA_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ACTLR_EL1, Auxiliary Control Register (EL1)

The ACTLR_EL1 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED configuration and control options for execution at EL1 and EL0.

Note

Arm recommends the contents of this register have no effect on the PE when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, and instead the configuration and control fields are provided by the ACTLR_EL2 register. This avoids the need for software to manage the contents of these register when switching between a Guest OS and a Host OS.

Configuration

AArch64 System register ACTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ACTLR[31:0].

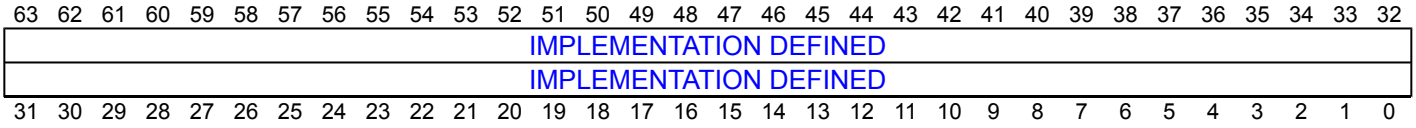
AArch64 System register ACTLR_EL1 bits [63:32] are architecturally mapped to AArch32 System register ACTLR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ACTLR_EL1 are UNDEFINED.

Attributes

ACTLR_EL1 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ACTLR_EL1

If the IMPLEMENTATION DEFINED ACTLR_EL12 accessor is implemented, the following behaviors are also implemented:

- MRS/MSR to ACTLR_EL1 from EL2 when the Effective value of HCR_EL2.E2H is 1 accesses ACTLR_EL2.
- MRS/MSR to ACTLR_EL1 from EL1 when the Effective value of HCR_EL2.{NV2, NV1, NV} is {1,0,1} accesses ACTLR_EL1 directly and is not transformed to a memory access.
- MRS/MSR to ACTLRALIAS_EL1 behaves in the same way as ACTLR_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ACTLR_EL1
```

op0	op1	CRn	CRm	op2

0b11	0b000	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TACR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (!boolean IMPLEMENTATION_DEFINED
"IMPLEMENTED_ACTLR_ELx accessor behavior" || EffectiveHCR_EL2_NVx() == '111') then
        X[t, 64] = NVMem[0x118];
    else
        X[t, 64] = ACTLR_EL1;
elsif PSTATE.EL == EL2 then
    if boolean IMPLEMENTATION_DEFINED "IMPLEMENTED_ACTLR_ELx accessor behavior" && ELIsInHost(EL2)
then
        X[t, 64] = ACTLR_EL2;
    else
        X[t, 64] = ACTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACTLR_EL1;
```

```
MSR ACTLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TACR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (!boolean IMPLEMENTATION_DEFINED
"IMPLEMENTED_ACTLR_ELx accessor behavior" || EffectiveHCR_EL2_NVx() == '111') then
        NVMem[0x118] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            ACTLR_EL1 = (X[t, 64] AND NOT EffectiveACTLRMASK_EL1()) OR (ACTLR_EL1 AND
EffectiveACTLRMASK_EL1());
        else
            ACTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if boolean IMPLEMENTATION_DEFINED "IMPLEMENTED_ACTLR_ELx accessor behavior" && ELIsInHost(EL2)
then
        if IsFeatureImplemented(FEAT_SRMASK) then
            ACTLR_EL2 = (X[t, 64] AND NOT EffectiveACTLRMASK_EL2()) OR (ACTLR_EL2 AND
EffectiveACTLRMASK_EL2());
        else
            ACTLR_EL2 = X[t, 64];
    else
        ACTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ACTLR_EL1 = X[t, 64];
```

When an implementation implements ACTLR_ELx accessor behavior and FEAT_VHE is implemented

```
MRS <Xt>, ACTLR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x118];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = ACTLR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = ACTLR_EL1;
    else
        UNDEFINED;
```

When an implementation implements ACTLR_ELx accessor behavior and FEAT_VHE is implemented

```
MSR ACTLR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x118] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        ACTLR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        ACTLR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_SRMASK is implemented

```
MRS <Xt>, ACTLRALIAS_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TACR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nACTLRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (!boolean IMPLEMENTATION_DEFINED
"IMPLEMENTED_ACTLR_ELx accessor behavior" || EffectiveHCR_EL2_NVx() == '111') then
        X[t, 64] = NVMem[0x118];
    else
        X[t, 64] = ACTLR_EL1;
elsif PSTATE.EL == EL2 then
    if boolean IMPLEMENTATION_DEFINED "IMPLEMENTED_ACTLR_ELx accessor behavior" && ELIsInHost(EL2)
then
        X[t, 64] = ACTLR_EL2;
    else
        X[t, 64] = ACTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACTLR_EL1;
```

When FEAT_SRMASK is implemented

```
MSR ACTLRALIAS_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TACR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nACTLRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (!boolean IMPLEMENTATION_DEFINED
"IMPLEMENTED_ACTLR_ELx accessor behavior" || EffectiveHCR_EL2_NVx() == '111') then
        NVMem[0x118] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            ACTLR_EL1 = (X[t, 64] AND NOT EffectiveACTLRMASK_EL1()) OR (ACTLR_EL1 AND
EffectiveACTLRMASK_EL1());
        else
            ACTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if boolean IMPLEMENTATION_DEFINED "IMPLEMENTED_ACTLR_ELx accessor behavior" && ELIsInHost(EL2)
then
        if IsFeatureImplemented(FEAT_SRMASK) then
            ACTLR_EL2 = (X[t, 64] AND NOT EffectiveACTLRMASK_EL2()) OR (ACTLR_EL2 AND
EffectiveACTLRMASK_EL2());
        else
            X[t, 64] = ACTLR_EL2;
    else
        ACTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ACTLR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ACTLR_EL2, Auxiliary Control Register (EL2)

The ACTLR_EL2 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED configuration and control options for EL2.

Note

Arm recommends the contents of this register are updated to apply to EL0 when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, gaining configuration and control fields from the ACTLR_EL1. This avoids the need for software to manage the contents of these register when switching between a Guest OS and a Host OS.

Configuration

AArch64 System register ACTLR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HACTLR[31:0].

AArch64 System register ACTLR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HACTLR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ACTLR_EL2 are UNDEFINED.

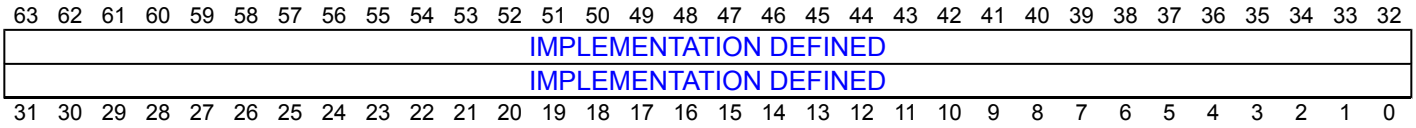
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ACTLR_EL2 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ACTLR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ACTLR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ACTLR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACTLR_EL2;
```

```
MSR ACTLR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_SRMASK) then
        ACTLR_EL2 = (X[t, 64] AND NOT EffectiveACTLRMASK_EL2()) OR (ACTLR_EL2 AND
EffectiveACTLRMASK_EL2());
    else
        ACTLR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ACTLR_EL2 = X[t, 64];
```

When an implementation implements ACTLR_ELx accessor behavior

```
MRS <Xt>, ACTLR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TACR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (!boolean IMPLEMENTATION_DEFINED
"IMPLEMENTED_ACTLR_ELx accessor behavior" || EffectiveHCR_EL2_NVx() == '111') then
        X[t, 64] = NVMem[0x118];
    else
        X[t, 64] = ACTLR_EL1;
elsif PSTATE.EL == EL2 then
    if boolean IMPLEMENTATION_DEFINED "IMPLEMENTED_ACTLR_ELx accessor behavior" && ELIsInHost(EL2)
then
        X[t, 64] = ACTLR_EL2;
    else
        X[t, 64] = ACTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACTLR_EL1;
```

When an implementation implements ACTLR_ELx accessor behavior

```
MSR ACTLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TACR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (!boolean IMPLEMENTATION_DEFINED
"IMPLEMENTED_ACTLR_ELx accessor behavior" || EffectiveHCR_EL2_NVx() == '111') then
        NVMem[0x118] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            ACTLR_EL1 = (X[t, 64] AND NOT EffectiveACTLRMASK_EL1()) OR (ACTLR_EL1 AND
EffectiveACTLRMASK_EL1());
        else
            ACTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if boolean IMPLEMENTATION_DEFINED "IMPLEMENTED_ACTLR_ELx accessor behavior" && ELIsInHost(EL2)
then
        if IsFeatureImplemented(FEAT_SRMASK) then
            ACTLR_EL2 = (X[t, 64] AND NOT EffectiveACTLRMASK_EL2()) OR (ACTLR_EL2 AND
EffectiveACTLRMASK_EL2());
        else
            ACTLR_EL2 = X[t, 64];
    else
        ACTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ACTLR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ACTLR_EL3, Auxiliary Control Register (EL3)

The ACTLR_EL3 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED configuration and control options for EL3.

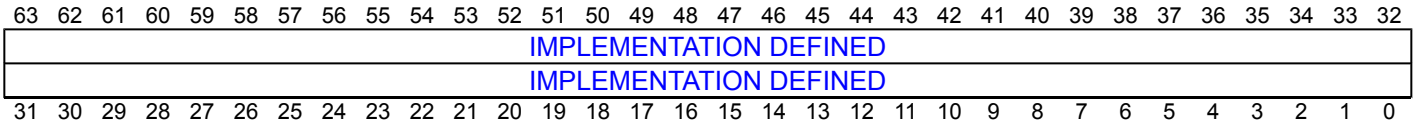
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ACTLR_EL3 are UNDEFINED.

Attributes

ACTLR_EL3 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ACTLR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ACTLR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0000	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACTLR_EL3;
```

```
MSR ACTLR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2

0b11	0b110	0b0001	0b0000	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.ACTLR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ACTLR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ACTLRMASK_EL1, Auxiliary Control Masking Register (EL1)

The ACTLRMASK_EL1 characteristics are:

Purpose

Mask register to prevent updates of fields in ACTLR_EL1 on writes to ACTLR_EL1 or ACTLRALIAS_EL1.

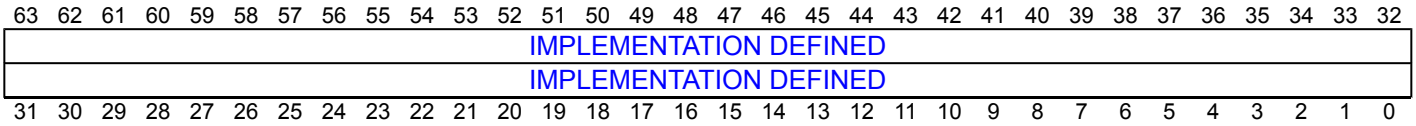
Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ACTLRMASK_EL1 are UNDEFINED.

Attributes

ACTLRMASK_EL1 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ACTLRMASK_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name ACTLRMASK_EL1 or ACTLRMASK_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ACTLRMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nACTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (!boolean IMPLEMENTATION_DEFINED
"IMPLEMENTED_ACTLR_ELx accessor behavior" || EffectiveHCR_EL2_NVx() == '111') then
        X[t, 64] = NVMem[0x340];
    else
        X[t, 64] = ACTLRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif boolean IMPLEMENTATION_DEFINED "IMPLEMENTED_ACTLR_ELx accessor behavior" &&
ELIsInHost(EL2) then
        X[t, 64] = ACTLRMASK_EL2;
    else
        X[t, 64] = ACTLRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACTLRMASK_EL1;
```

```
MSR ACTLRMASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nACTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (!boolean IMPLEMENTATION_DEFINED
"IMPLEMENTED_ACTLR_ELx accessor behavior" || EffectiveHCR_EL2_NVx() == '111') then
        NVMem[0x340] = X[t, 64];
    elsif !IsZero(EffectiveACTLRMASK_EL1()) then
        UNDEFINED;
    else
        ACTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif boolean IMPLEMENTATION_DEFINED "IMPLEMENTED_ACTLR_ELx accessor behavior" &&
ELIsInHost(EL2) then
        if !IsZero(EffectiveACTLRMASK_EL2()) then
            UNDEFINED;
        else
            ACTLRMASK_EL2 = X[t, 64];
    else
        ACTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ACTLRMASK_EL1 = X[t, 64];
```

When an implementation implements ACTLR_ELx accessor behavior and FEAT_VHE is implemented

```
MRS <Xt>, ACTLRMASK_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x340];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = ACTLRMASK_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = ACTLRMASK_EL1;
    else
        UNDEFINED;
```

When an implementation implements ACTLR_ELx accessor behavior and FEAT_VHE is implemented

```
MSR ACTLRMASK_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x340] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ACTLRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        ACTLRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ACTLRMASK_EL2, Auxiliary Control Masking Register (EL2)

The ACTLRMASK_EL2 characteristics are:

Purpose

Mask register to prevent updates of fields in ACTLR_EL2 on writes.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ACTLRMASK_EL2 are UNDEFINED.

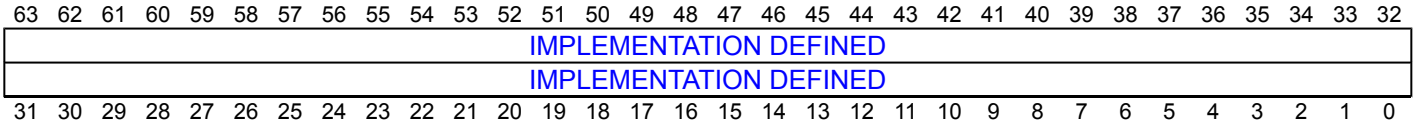
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ACTLRMASK_EL2 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ACTLRMASK_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name ACTLRMASK_EL2 or ACTLRMASK_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ACTLRMASK_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ACTLRMASK_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACTLRMASK_EL2;
```

```
MSR ACTLRMASK_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsZero(EffectiveACTLRMASK_EL2()) then
        UNDEFINED;
    else
        ACTLRMASK_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ACTLRMASK_EL2 = X[t, 64];
```

```
MRS <Xt>, ACTLRMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nACTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (!boolean IMPLEMENTATION_DEFINED
"IMPLEMENTED_ACTLR_ELx accessor behavior" || EffectiveHCR_EL2_NVx() == '111') then
        X[t, 64] = NVMem[0x340];
    else
        X[t, 64] = ACTLRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif boolean IMPLEMENTATION_DEFINED "IMPLEMENTED_ACTLR_ELx accessor behavior" &&
ELIsInHost(EL2) then
        X[t, 64] = ACTLRMASK_EL2;
    else
        X[t, 64] = ACTLRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ACTLRMASK_EL1;
```

```
MSR ACTLRMASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nACTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (!boolean IMPLEMENTATION_DEFINED
"IMPLEMENTED_ACTLR_ELx accessor behavior" || EffectiveHCR_EL2_NVx() == '111') then
        NVMem[0x340] = X[t, 64];
    elsif !IsZero(EffectiveACTLRMASK_EL1()) then
        UNDEFINED;
    else
        ACTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif boolean IMPLEMENTATION_DEFINED "IMPLEMENTED_ACTLR_ELx accessor behavior" &&
ELIsInHost(EL2) then
        if !IsZero(EffectiveACTLRMASK_EL2()) then
            UNDEFINED;
        else
            ACTLRMASK_EL2 = X[t, 64];
    else
        ACTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ACTLRMASK_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR0_EL1, Auxiliary Fault Status Register 0 (EL1)

The AFSR0_EL1 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL1.

Configuration

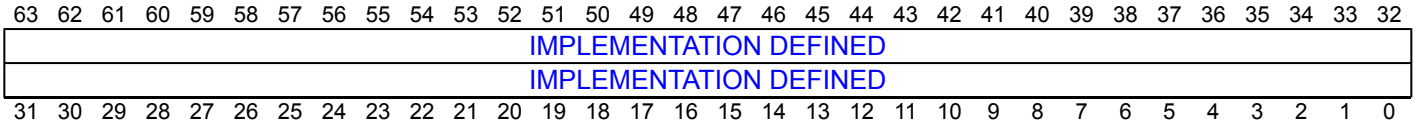
AArch64 System register AFSR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ADFSR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AFSR0_EL1 are UNDEFINED.

Attributes

AFSR0_EL1 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR0_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name AFSR0_EL1 or AFSR0_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AFSR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.AFSR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x128];
    else
        X[t, 64] = AFSR0_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = AFSR0_EL2;
    else
        X[t, 64] = AFSR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AFSR0_EL1;
```

```
MSR AFSR0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.AFSR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x128] = X[t, 64];
    else
        AFSR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AFSR0_EL2 = X[t, 64];
    else
        AFSR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AFSR0_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, AFSR0_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x128];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = AFSR0_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = AFSR0_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR AFSR0_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x128] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AFSR0_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        AFSR0_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR0_EL2, Auxiliary Fault Status Register 0 (EL2)

The AFSR0_EL2 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL2.

Configuration

AArch64 System register AFSR0_EL2 bits [31:0] are architecturally mapped to AArch32 System register HADFSR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AFSR0_EL2 are UNDEFINED.

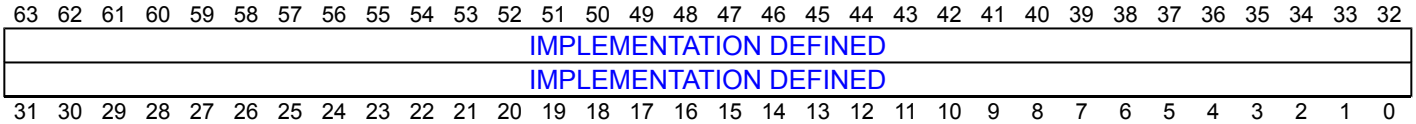
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

AFSR0_EL2 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR0_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name AFSR0_EL2 or AFSR0_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AFSR0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = AFSR0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AFSR0_EL2;
```

```
MSR AFSR0_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AFSR0_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AFSR0_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, AFSR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.AFSR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x128];
    else
        X[t, 64] = AFSR0_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = AFSR0_EL2;
    else
        X[t, 64] = AFSR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AFSR0_EL1;
```

When FEAT_VHE is implemented

```
MSR AFSR0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.AFSR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x128] = X[t, 64];
    else
        AFSR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AFSR0_EL2 = X[t, 64];
    else
        AFSR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AFSR0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR0_EL3, Auxiliary Fault Status Register 0 (EL3)

The AFSR0_EL3 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL3.

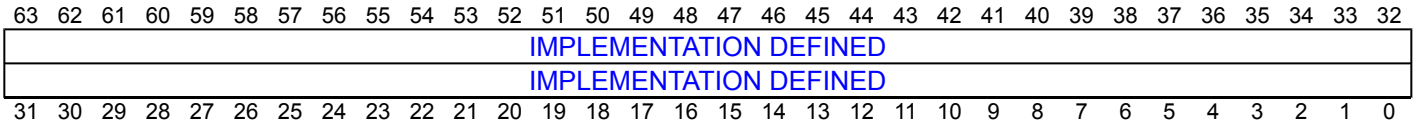
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AFSR0_EL3 are UNDEFINED.

Attributes

AFSR0_EL3 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR0_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AFSR0_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0001	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AFSR0_EL3;
```

```
MSR AFSR0_EL3, <Xt>
```

op0	op1	CRn	CRm	op2

0b11	0b110	0b0101	0b0001	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.AFSR0_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AFSR0_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR1_EL1, Auxiliary Fault Status Register 1 (EL1)

The AFSR1_EL1 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL1.

Configuration

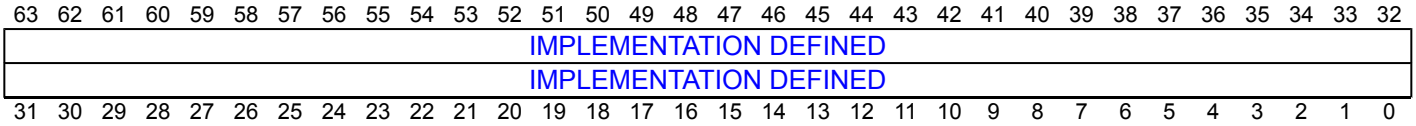
AArch64 System register AFSR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register AIFSR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AFSR1_EL1 are UNDEFINED.

Attributes

AFSR1_EL1 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR1_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name AFSR1_EL1 or AFSR1_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AFSR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.AFSR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x130];
    else
        X[t, 64] = AFSR1_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = AFSR1_EL2;
    else
        X[t, 64] = AFSR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AFSR1_EL1;
```

```
MSR AFSR1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.AFSR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x130] = X[t, 64];
    else
        AFSR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AFSR1_EL2 = X[t, 64];
    else
        AFSR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AFSR1_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, AFSR1_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x130];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = AFSR1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = AFSR1_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR AFSR1_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x130] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AFSR1_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        AFSR1_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR1_EL2, Auxiliary Fault Status Register 1 (EL2)

The AFSR1_EL2 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL2.

Configuration

AArch64 System register AFSR1_EL2 bits [31:0] are architecturally mapped to AArch32 System register HAIFSR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AFSR1_EL2 are UNDEFINED.

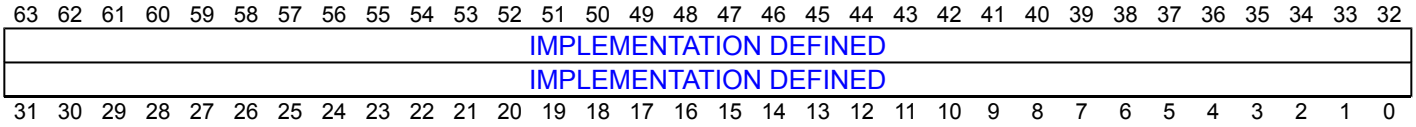
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

AFSR1_EL2 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR1_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name AFSR1_EL2 or AFSR1_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AFSR1_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = AFSR1_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AFSR1_EL2;
```

```
MSR AFSR1_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AFSR1_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AFSR1_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, AFSR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.AFSR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x130];
    else
        X[t, 64] = AFSR1_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = AFSR1_EL2;
    else
        X[t, 64] = AFSR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AFSR1_EL1;
```

When FEAT_VHE is implemented

```
MSR AFSR1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.AFSR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x130] = X[t, 64];
    else
        AFSR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AFSR1_EL2 = X[t, 64];
    else
        AFSR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AFSR1_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR1_EL3, Auxiliary Fault Status Register 1 (EL3)

The AFSR1_EL3 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL3.

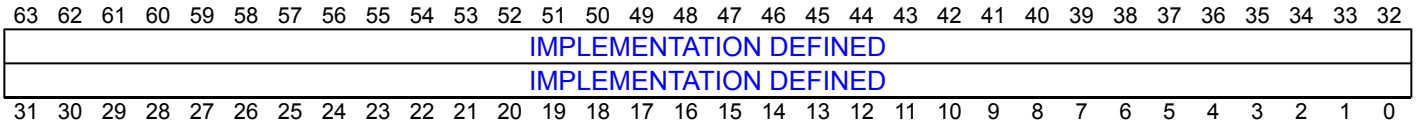
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AFSR1_EL3 are UNDEFINED.

Attributes

AFSR1_EL3 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR1_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AFSR1_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0001	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AFSR1_EL3;
```

```
MSR AFSR1_EL3, <Xt>
```

op0	op1	CRn	CRm	op2

0b11	0b110	0b0101	0b0001	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.AFSR1_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AFSR1_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AIDR_EL1, Auxiliary ID Register

The AIDR_EL1 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED identification information.

The value of this register must be interpreted in conjunction with the value of MIDR_EL1.

Configuration

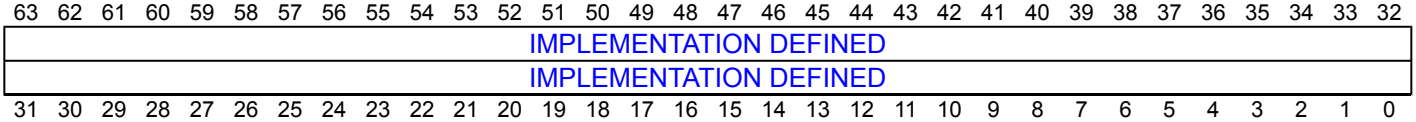
AArch64 System register AIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register AIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AIDR_EL1 are UNDEFINED.

Attributes

AIDR_EL1 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing AIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b001	0b0000	0b0000	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.AIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = AIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = AIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ALLINT, All Interrupt Mask Bit

The ALLINT characteristics are:

Purpose

Allows access to the all interrupt mask bit.

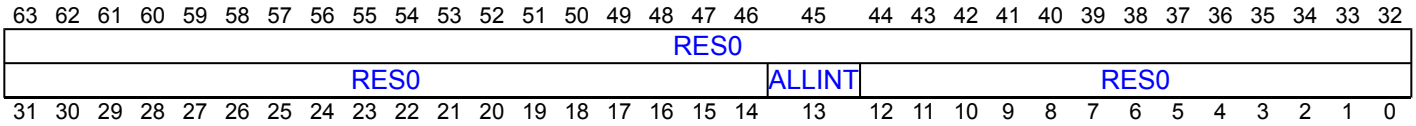
Configuration

This register is present only when FEAT_NMI is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ALLINT are UNDEFINED.

Attributes

ALLINT is a 64-bit register.

Field descriptions



Bits [63:14]

Reserved, RES0.

ALLINT, bit [13]

All interrupt mask. An interrupt is controlled by PSTATE.ALLINT when all of the following apply:

- SCTLR_ELx.NMI is 1.
- The interrupt is targeted at ELx.
- Execution is at ELx.

ALLINT	Meaning
0b0	This control does not cause any interrupts to be masked.
0b1	If SCTLR_ELx.NMI is 1 and execution is at ELx, an IRQ or FIQ interrupt that is targeted to ELx, with or without Superpriority, is masked.

The value of this bit is set to the inverse value in the SCTLR_ELx.SPINTMASK field on taking an exception to ELx.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [12:0]

Reserved, RES0.

Accessing ALLINT

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ALLINT

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_NMI) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    X[t, 64] = Zeros(50):PSTATE.ALLINT:Zeros(13);
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(50):PSTATE.ALLINT:Zeros(13);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(50):PSTATE.ALLINT:Zeros(13);
```

MSR ALLINT, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_NMI) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsHCRXEL2Enabled() && HCRX_EL2.TALLINT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        PSTATE.ALLINT = X[t, 64]<13>;
elsif PSTATE.EL == EL2 then
    PSTATE.ALLINT = X[t, 64]<13>;
elsif PSTATE.EL == EL3 then
    PSTATE.ALLINT = X[t, 64]<13>;
```

MSR ALLINT, #<imm>

op0	op1	CRn	CRm	op2
0b00	0b001	0b0100	0b000x	0b000

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR2_EL1, Extended Auxiliary Memory Attribute Indirection Register (EL1)

The AMAIR2_EL1 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR2_EL1.

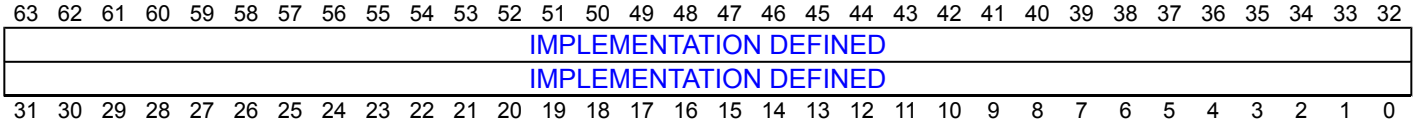
Configuration

This register is present only when FEAT_AIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AMAIR2_EL1 are UNDEFINED.

Attributes

AMAIR2_EL1 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIR2_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name AMAIR2_EL1 or AMAIR2_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMAIR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nAMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x288];
    else
        X[t, 64] = AMAIR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = AMAIR2_EL2;
    else
        X[t, 64] = AMAIR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMAIR2_EL1;
```

```
MSR AMAIR2_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nAMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x288] = X[t, 64];
    else
        AMAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        AMAIR2_EL2 = X[t, 64];
    else
        AMAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMAIR2_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, AMAIR2_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x288];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = AMAIR2_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = AMAIR2_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR AMAIR2_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x288] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AMAIR2_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        AMAIR2_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR2_EL2, Extended Auxiliary Memory Attribute Indirection Register (EL2)

The AMAIR2_EL2 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR2_EL2.

Configuration

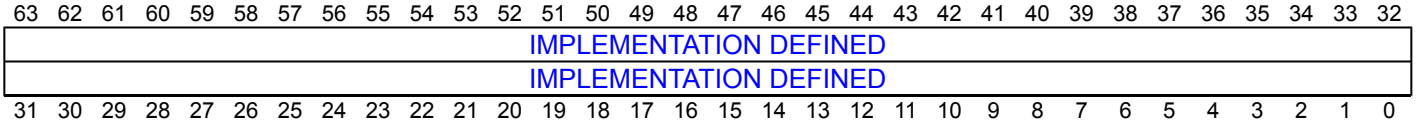
This register is present only when FEAT_AIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AMAIR2_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

AMAIR2_EL2 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing AMAIR2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name AMAIR2_EL2 or AMAIR2_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMAIR2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMAIR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMAIR2_EL2;
```

MSR AMAIR2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMAIR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMAIR2_EL2 = X[t, 64];
```

MRS <Xt>, AMAIR2_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nAMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x288];
    else
        X[t, 64] = AMAIR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = AMAIR2_EL2;
    else
        X[t, 64] = AMAIR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMAIR2_EL1;
```

```
MSR AMAIR2_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nAMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x288] = X[t, 64];
    else
        AMAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        AMAIR2_EL2 = X[t, 64];
    else
        AMAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMAIR2_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR2_EL3, Extended Auxiliary Memory Attribute Indirection Register (EL3)

The AMAIR2_EL3 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR2_EL3.

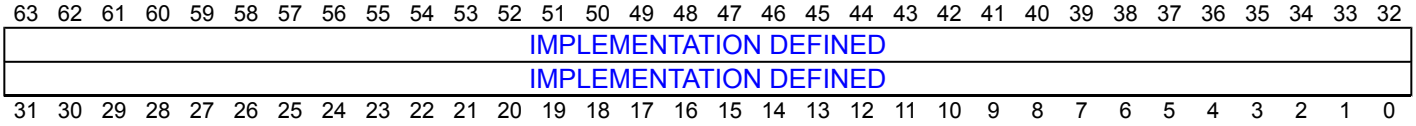
Configuration

This register is present only when FEAT_AIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AMAIR2_EL3 are UNDEFINED.

Attributes

AMAIR2_EL3 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing AMAIR2_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMAIR2_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMAIR2_EL3;
```

```
MSR AMAIR2_EL3, <Xt>
```

op0	op1	CRn	CRm	op2

0b11	0b110	0b1010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.AMAIR2_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMAIR2_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR_EL1, Auxiliary Memory Attribute Indirection Register (EL1)

The AMAIR_EL1 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL1.

Configuration

AArch64 System register AMAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register AMAIR0[31:0].

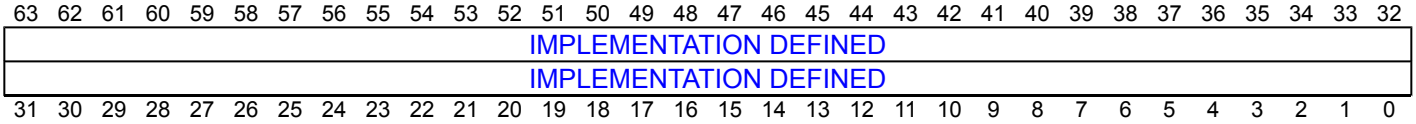
AArch64 System register AMAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register AMAIR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AMAIR_EL1 are UNDEFINED.

Attributes

AMAIR_EL1 is a 64-bit register.

Field descriptions



AMAIR_EL1 is permitted to be cached in a TLB.

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name AMAIR_EL1 or AMAIR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMAIR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.AMAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x148];
    else
        X[t, 64] = AMAIR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = AMAIR_EL2;
    else
        X[t, 64] = AMAIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMAIR_EL1;
```

```
MSR AMAIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.AMAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x148] = X[t, 64];
    else
        AMAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AMAIR_EL2 = X[t, 64];
    else
        AMAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMAIR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, AMAIR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x148];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = AMAIR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = AMAIR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR AMAIR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x148] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AMAIR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        AMAIR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR_EL2, Auxiliary Memory Attribute Indirection Register (EL2)

The AMAIR_EL2 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL2.

Configuration

AArch64 System register AMAIR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HAMAIR0[31:0].

AArch64 System register AMAIR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HAMAIR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AMAIR_EL2 are UNDEFINED.

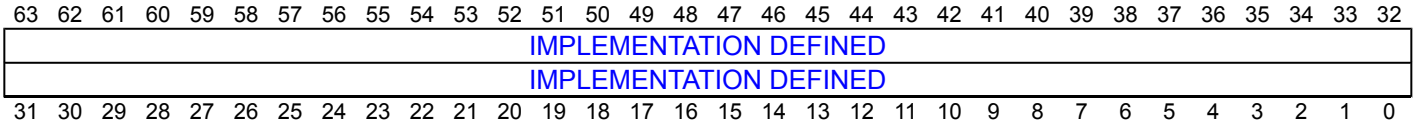
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

AMAIR_EL2 is a 64-bit register.

Field descriptions



AMAIR_EL2 is permitted to be cached in a TLB.

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name AMAIR_EL2 or AMAIR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMAIR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = AMAIR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMAIR_EL2;
```

```
MSR AMAIR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AMAIR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMAIR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, AMAIR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.AMAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x148];
    else
        X[t, 64] = AMAIR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = AMAIR_EL2;
    else
        X[t, 64] = AMAIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMAIR_EL1;
```

When FEAT_VHE is implemented

```
MSR AMAIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.AMAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x148] = X[t, 64];
    else
        AMAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AMAIR_EL2 = X[t, 64];
    else
        AMAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMAIR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR_EL3, Auxiliary Memory Attribute Indirection Register (EL3)

The AMAIR_EL3 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL3.

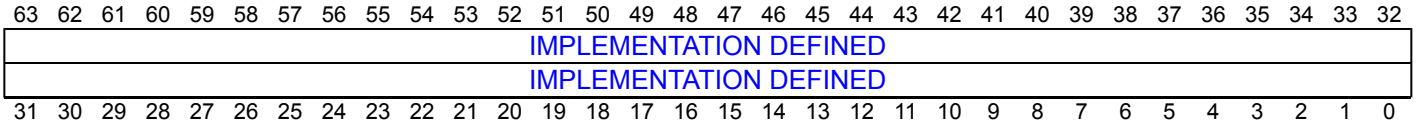
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AMAIR_EL3 are UNDEFINED.

Attributes

AMAIR_EL3 is a 64-bit register.

Field descriptions



AMAIR_EL3 is permitted to be cached in a TLB.

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMAIR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0011	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMAIR_EL3;
```

```
MSR AMAIR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0011	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.AMAIR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMAIR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCFGR_EL0, Activity Monitors Configuration Register

The AMCFGR_EL0 characteristics are:

Purpose

Global configuration register for the activity monitors.

Provides information on supported features, the number of counter groups implemented, the total number of activity monitor event counters implemented, and the size of the counters. AMCFGR_EL0 is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch64 System register AMCFGR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCFGR[31:0].

AArch64 System register AMCFGR_EL0 bits [31:0] are architecturally mapped to External register AMCFGR[31:0] when FEAT_AMU_EXT32 is implemented.

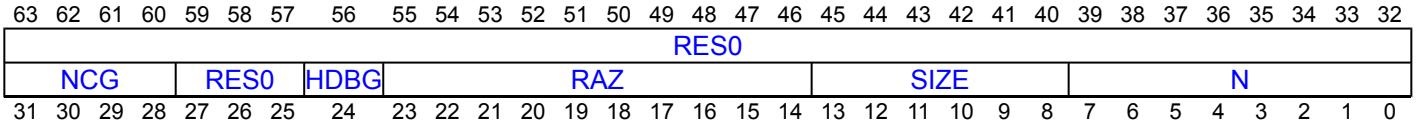
AArch64 System register AMCFGR_EL0 bits [63:0] are architecturally mapped to External register AMCFGR[63:0] when FEAT_AMU_EXT64 is implemented.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCFGR_EL0 are UNDEFINED.

Attributes

AMCFGR_EL0 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NCG, bits [31:28]

Defines the number of counter groups implemented, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCG	Meaning
0b0000	One counter group implemented.
0b0001	Two counter groups implemented.

All other values are reserved.

Access to this field is **RO**.

Bits [27:25]

Reserved, RES0.

HDBG, bit [24]

Halt-on-debug supported.

This feature must be supported, and so this bit is `0b1`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HDBG	Meaning
`0b0`	AMCR_EL0.HDBG is RES0.
`0b1`	AMCR_EL0.HDBG is read/write.

Access to this field is **RO**.

Bits [23:14]

Reserved, RAZ.

SIZE, bits [13:8]

Defines the size of the activity monitor event counters, minus one.

The counters are 64-bit, so the value of this field is `0b111111`.

This field is used by software to determine the spacing of the counters in the memory-map. The counters are at doubleword-aligned addresses.

Reads as `0b111111`.

Access to this field is **RO**.

N, bits [7:0]

Defines the number of activity monitor event counters implemented in all groups, minus one.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMCFGR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCFGR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCFGR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCFGR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCFGR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCFGR_EL0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCG1IDR_EL0, Activity Monitors Counter Group 1 Identification Register

The AMCG1IDR_EL0 characteristics are:

Purpose

Defines which auxiliary counters are implemented, and which of them have a corresponding virtual offset register, AMEVCNTVOFF1<n>_EL2 implemented.

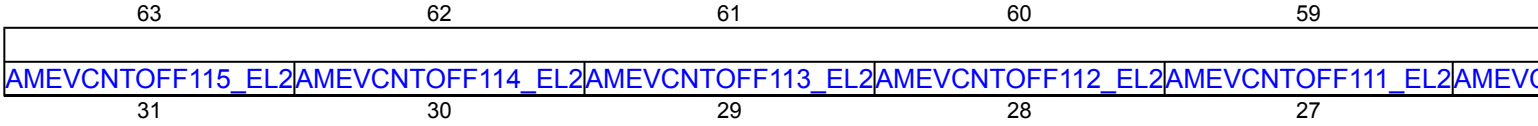
Configuration

This register is present only when FEAT_AMUv1p1 is implemented. Otherwise, direct accesses to AMCG1IDR_EL0 are UNDEFINED.

Attributes

AMCG1IDR_EL0 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

AMEVCNTOFF1<n>_EL2, bit [n+16], for n = 15 to 0

Indicates which implemented auxiliary counters have a corresponding virtual offset register, AMEVCNTVOFF1<n>_EL2 implemented.

AMEVCNTOFF1<n>_EL2	Meaning
0b0	AMEVCNTR1<n>_EL0 does not have an offset, or is not implemented.
0b1	The offset AMEVCNTVOFF1<n>_EL2 is implemented for AMEVCNTR1<n>_EL0.

AMEVCNTR1<n>_EL0, bit [n], for n = 15 to 0

Indicates which auxiliary counters AMEVCNTR1<n>_EL0 are implemented.

AMEVCNTR1<n>_EL0	Meaning
0b0	AMEVCNTR1<n>_EL0 is not implemented.
0b1	AMEVCNTR1<n>_EL0 is implemented.

Accessing AMCG1IDR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCG1IDR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_AMUv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCG1IDR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCG1IDR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCG1IDR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCG1IDR_EL0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCGCR_EL0, Activity Monitors Counter Group Configuration Register

The AMCGCR_EL0 characteristics are:

Purpose

Provides information on the number of activity monitor event counters implemented within each counter group.

Configuration

AArch64 System register AMCGCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCGCR[31:0].

AArch64 System register AMCGCR_EL0 bits [31:0] are architecturally mapped to External register AMCGCR[31:0] when FEAT_AMU_EXT32 is implemented.

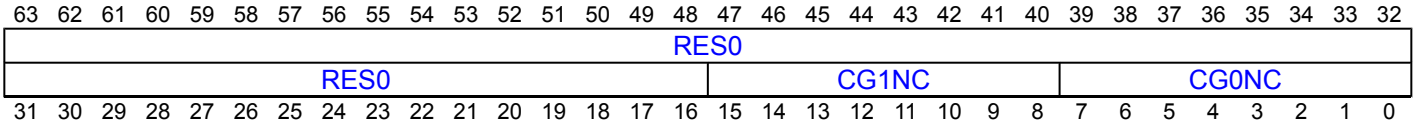
AArch64 System register AMCGCR_EL0 bits [63:0] are architecturally mapped to External register AMCGCR[63:0] when FEAT_AMU_EXT64 is implemented.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCGCR_EL0 are UNDEFINED.

Attributes

AMCGCR_EL0 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

CG1NC, bits [15:8]

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CG1NC	Meaning
0x00..0x10	The number of counters.

All other values are reserved.

Access to this field is **RO**.

CG0NC, bits [7:0]

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

Reads as 0x04.

Access to this field is **RO**.

Accessing AMCGCR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCGCR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCGCR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCGCR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCGCR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCGCR_EL0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENCLR0_EL0, Activity Monitors Count Enable Clear Register 0

The AMCNTENCLR0_EL0 characteristics are:

Purpose

Disable control bits for the architected activity monitors event counters, AMEVCNTR0<n>_EL0.

Configuration

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR0[31:0].

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR0[31:0].

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR[31:0].

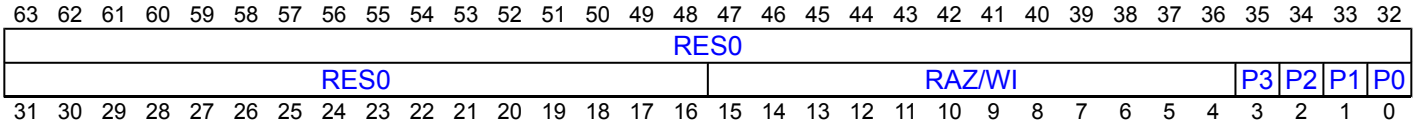
AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET0[31:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR0_EL0 are UNDEFINED.

Attributes

AMCNTENCLR0_EL0 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter disable bit for AMEVCNTR0<n>_EL0.

Note

AMCGCR_EL0.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR0<n>_EL0 is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR0<n>_EL0 is enabled. When written, disables AMEVCNTR0<n>_EL0.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= 4, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing AMCNTENCLR0_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCNTENCLR0_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b100

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR0_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR0_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR0_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCNTENCLR0_EL0;
```

```
MSR AMCNTENCLR0_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b100

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENCLR0_EL0 = X[t, 64];
else
    UNDEFINED;
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENCLR1_EL0, Activity Monitors Count Enable Clear Register 1

The AMCNTENCLR1_EL0 characteristics are:

Purpose

Disable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>_EL0.

Configuration

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR1[31:0].

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR1[31:0].

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET1[31:0].

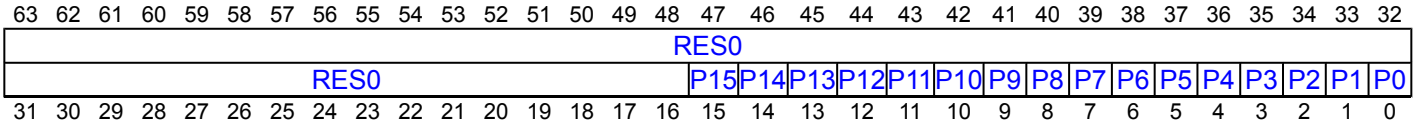
AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR[63:32].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR1_EL0 are UNDEFINED.

Attributes

AMCNTENCLR1_EL0 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter disable bit for AMEVCNTR1<n>_EL0.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR1<n>_EL0 is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR1<n>_EL0 is enabled. When written, disables AMEVCNTR1<n>_EL0.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= UInt(AMCGCR_EL0.CG1NC), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing AMCNTENCLR1_EL0

If there are no auxiliary monitor event counters implemented, reads and writes of AMCNTENCLR1_EL0 are UNDEFINED.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR_EL0.NCG == 0b0000.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCNTENCLR1_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR1_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR1_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR1_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCNTENCLR1_EL0;
```

```
MSR AMCNTENCLR1_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENCLR1_EL0 = X[t, 64];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET0_EL0, Activity Monitors Count Enable Set Register 0

The AMCNTENSET0_EL0 characteristics are:

Purpose

Enable control bits for the architected activity monitors event counters, AMEVCNTR0<n>_EL0.

Configuration

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET0[31:0].

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET0[31:0].

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET[31:0].

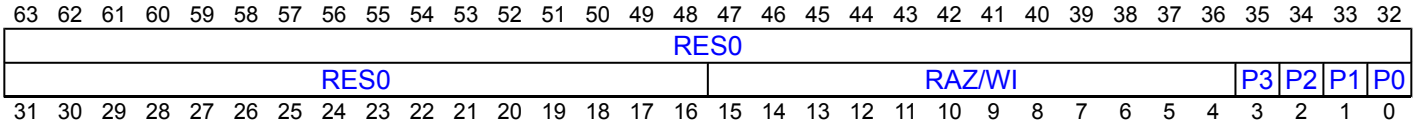
AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR0[31:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET0_EL0 are UNDEFINED.

Attributes

AMCNTENSET0_EL0 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter enable bit for AMEVCNTR0<n>_EL0.

Note

AMCGCR_EL0.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR0<n>_EL0 is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR0<n>_EL0 is enabled. When written, enables AMEVCNTR0<n>_EL0.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= 4, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing AMCNTENSET0_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCNTENSET0_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b101

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET0_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET0_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET0_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCNTENSET0_EL0;
```

```
MSR AMCNTENSET0_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b101

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENSET0_EL0 = X[t, 64];
else
    UNDEFINED;
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET1_EL0, Activity Monitors Count Enable Set Register 1

The AMCNTENSET1_EL0 characteristics are:

Purpose

Enable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>_EL0.

Configuration

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET1[31:0].

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET1[31:0].

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR1[31:0].

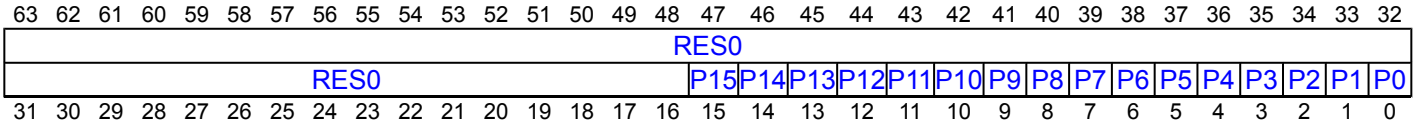
AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET[63:32].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET1_EL0 are UNDEFINED.

Attributes

AMCNTENSET1_EL0 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter enable bit for AMEVCNTR1<n>_EL0.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR1<n>_EL0 is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR1<n>_EL0 is enabled. When written, enables AMEVCNTR1<n>_EL0.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= UInt(AMCGCR_EL0.CG1NC), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing AMCNTENSET1_EL0

If there are no auxiliary monitor event counters implemented, reads and writes of AMCNTENSET1_EL0 are UNDEFINED.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR_EL0.NCG == 0b0000.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCNTENSET1_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET1_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET1_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET1_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCNTENSET1_EL0;
```

```
MSR AMCNTENSET1_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENSET1_EL0 = X[t, 64];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCR_EL0, Activity Monitors Control Register

The AMCR_EL0 characteristics are:

Purpose

Global control register for the activity monitors implementation. AMCR_EL0 is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch64 System register AMCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCR[31:0].

AArch64 System register AMCR_EL0 bits [31:0] are architecturally mapped to External register AMCR[31:0] when FEAT_AMU_EXT32 is implemented.

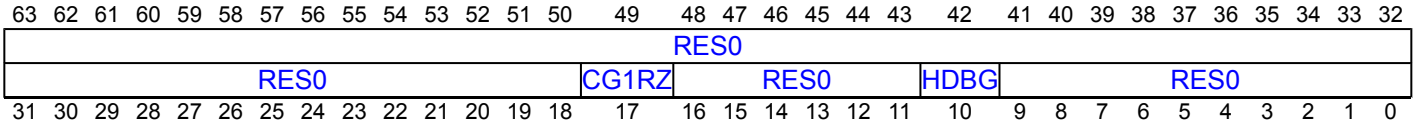
AArch64 System register AMCR_EL0 bits [63:0] are architecturally mapped to External register AMCR[63:0] when FEAT_AMU_EXT64 is implemented.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCR_EL0 are UNDEFINED.

Attributes

AMCR_EL0 is a 64-bit register.

Field descriptions



Bits [63:18]

Reserved, RES0.

CG1RZ, bit [17]

When FEAT_AMUv1p1 is implemented:

Counter Group 1 Read Zero.

CG1RZ	Meaning
0b0	System register reads of AMEVCNTR1<n>_EL0 return the event count at all implemented and enabled Exception levels.
0b1	If the current Exception level is the highest implemented Exception level, system register reads of AMEVCNTR1<n>_EL0 return the event count. Otherwise, reads of AMEVCNTR1<n>_EL0 return a zero value.

Note

Reads from the memory-mapped view are unaffected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:11]

Reserved, RES0.

HDBG, bit [10]

This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

HDBG	Meaning
0b0	Activity monitors do not halt counting when the PE is halted in Debug state.
0b1	Activity monitors halt counting when the PE is halted in Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing AMCR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCR_EL0;
```

```
MSR AMCR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMCR_EL0 = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTR0<n>_EL0, Activity Monitors Event Counter Registers 0, n = 0 - 3

The AMEVCNTR0<n>_EL0 characteristics are:

Purpose

Provides access to the architected activity monitor event counters.

Configuration

AArch64 System register AMEVCNTR0<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVCNTR0<n>[31:0].

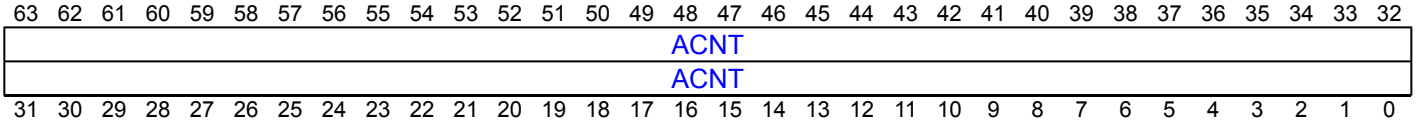
AArch64 System register AMEVCNTR0<n>_EL0 bits [63:0] are architecturally mapped to External register AMEVCNTR0<n>[63:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<n>_EL0 are UNDEFINED.

Attributes

AMEVCNTR0<n>_EL0 is a 64-bit register.

Field descriptions



ACNT, bits [63:0]

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 3.

If all of the following are true, reads of the AMEVCNTR0<n>_EL0 registers from EL0 or EL1 return (PCount<63:0> - AMEVCNTVOFF0<n>_EL2<63:0>), where PCount is the physical count returned when AMEVCNTR0<n>_EL0 is read from EL2 or EL3:

- FEAT_AMUv1p1 is implemented.
- HCR_EL2.AMVOFFEN and SCR_EL3.AMVOFFEN are 1.
- EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0000000000000000000000000000000000000000000000000000000000000000'.

Accessing AMEVCNTR0<n>_EL0

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVCNTR0<n>_EL0 are UNDEFINED.

Note

AMCGCR_EL0.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVCNTR0<m>_EL0 ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b010:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR0<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVCNTR0_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HAFGRTR_EL2.AMEVCNTR0<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVCNTR0_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVCNTR0_EL0[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVCNTR0_EL0[m];
```

```
MSR AMEVCNTR0<m>_EL0, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b010:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMEVCNTR0_EL0[m] = X[t, 64];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTR1<n>_EL0, Activity Monitors Event Counter Registers 1, n = 0 - 15

The AMEVCNTR1<n>_EL0 characteristics are:

Purpose

Provides access to the auxiliary activity monitor event counters.

Configuration

AArch64 System register AMEVCNTR1<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVCNTR1<n>[31:0].

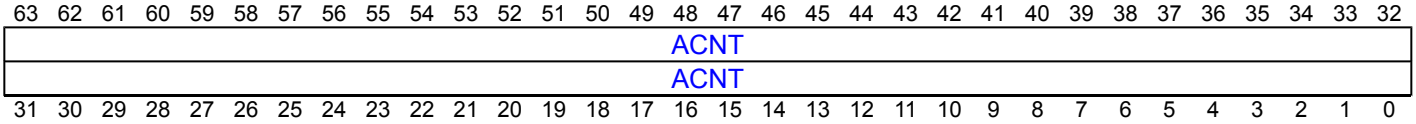
AArch64 System register AMEVCNTR1<n>_EL0 bits [63:0] are architecturally mapped to External register AMEVCNTR1<n>[63:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n>_EL0 are UNDEFINED.

Attributes

AMEVCNTR1<n>_EL0 is a 64-bit register.

Field descriptions



ACNT, bits [63:0]

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If all of the following are true, reads of the AMEVCNTR1<n>_EL0 registers from EL0 or EL1 return (PCount<63:0> - AMEVCNTVOFF1<n>_EL2<63:0>), where PCount is the physical count returned when AMEVCNTR1<n>_EL0 is read from EL2 or EL3:

- FEAT_AMUv1p1 is implemented.
- HCR_EL2.AMVOFFEN and SCR_EL3.AMVOFFEN are 1.
- AMCR_EL0.CG1RZ is 0.
- EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0000000000000000000000000000000000000000000000000000000000000000'.

Accessing AMEVCNTR1<n>_EL0

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVCNTR1<n>_EL0 are UNDEFINED.

Note

AMCGCR_EL0.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVCNTR1<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b110:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR1<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif AMCR_EL0.CG1RZ == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = AMEVCNTR1_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HAFGRTR_EL2.AMEVCNTR1<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsHighestEL(PSTATE.EL) && AMCR_EL0.CG1RZ == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = AMEVCNTR1_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsHighestEL(PSTATE.EL) && AMCR_EL0.CG1RZ == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = AMEVCNTR1_EL0[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVCNTR1_EL0[m];
```

```
MSR AMEVCNTR1<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b110:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMEVCNTR1_EL0[m] = X[t, 64];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTVOFF0<n>_EL2, Activity Monitors Event Counter Virtual Offset Registers 0, n = 0 - 15

The AMEVCNTVOFF0<n>_EL2 characteristics are:

Purpose

Holds the 64-bit virtual offset for architected activity monitor events.

Configuration

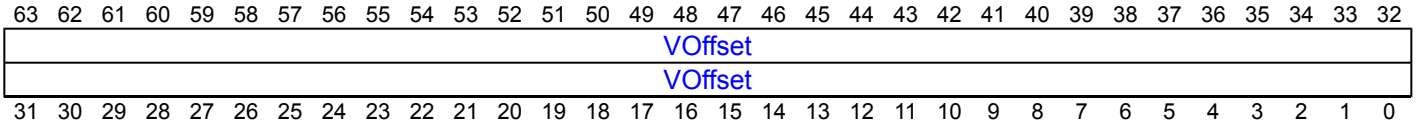
This register is present only when FEAT_AMUv1p1 is implemented. Otherwise, direct accesses to AMEVCNTVOFF0<n>_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

AMEVCNTVOFF0<n>_EL2 is a 64-bit register.

Field descriptions



VOffset, bits [63:0]

Virtual offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVCNTVOFF0<n>_EL2

If <n> is not 0, 2 or 3, reads and writes of AMEVCNTVOFF0<n>_EL2 are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVCNTVOFF0<m>_EL2 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b100:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1p1) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif !(m IN {0, 2, 3}) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0xA00 + (8 * m)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AMVOFFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMVOFFEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVCNTVOFF0_EL2[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVCNTVOFF0_EL2[m];
```

```
MSR AMEVCNTVOFF0<m>_EL2, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b100:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1p1) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif !(m IN {0, 2, 3}) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0xA00 + (8 * m)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AMVOFFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMVOFFEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMEVCNTVOFF0_EL2[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMEVCNTVOFF0_EL2[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTVOFF1<n>_EL2, Activity Monitors Event Counter Virtual Offset Registers 1, n = 0 - 15

The AMEVCNTVOFF1<n>_EL2 characteristics are:

Purpose

Holds the 64-bit virtual offset for auxiliary activity monitor events.

Configuration

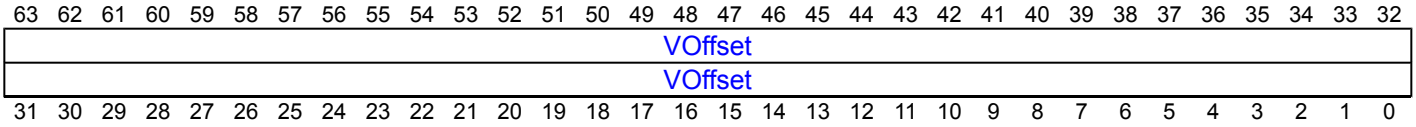
This register is present only when FEAT_AMUv1p1 is implemented. Otherwise, direct accesses to AMEVCNTVOFF1<n>_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

AMEVCNTVOFF1<n>_EL2 is a 64-bit register.

Field descriptions



VOffset, bits [63:0]

Virtual offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVCNTVOFF1<n>_EL2

Note

AMCG1IDR_EL0 identifies which auxiliary activity monitor event counters have a corresponding virtual offset implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVCNTVOFF1<m>_EL2 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b101:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1p1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorOffsetImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0xA80 + (8 * m)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AMVOFFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMVOFFEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVCNTVOFF1_EL2[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVCNTVOFF1_EL2[m];
```

```
MSR AMEVCNTVOFF1<m>_EL2, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b101:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1p1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorOffsetImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0xA80 + (8 * m)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AMVOFFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMVOFFEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMEVCNTVOFF1_EL2[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMEVCNTVOFF1_EL2[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVTYPER0<n>_EL0, Activity Monitors Event Type Registers 0, n = 0 - 3

The AMEVTYPER0<n>_EL0 characteristics are:

Purpose

Provides information on the events that an architected activity monitor event counter AMEVCNTR0<n>_EL0 counts.

Configuration

AArch64 System register AMEVTYPER0<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER0<n>[31:0].

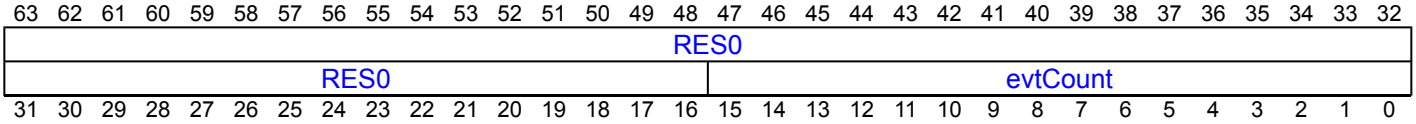
AArch64 System register AMEVTYPER0<n>_EL0 bits [31:0] are architecturally mapped to External register AMEVTYPER0<n>[31:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n>_EL0 are UNDEFINED.

Attributes

AMEVTYPER0<n>_EL0 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>_EL0. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

evtCount	Meaning	Applies when
0x0011	Processor frequency cycles.	When n == 0
0x4004	Constant frequency cycles.	When n == 1
0x0008	Instructions retired.	When n == 2
0x4005	Memory stall cycles.	When n == 3

Access to this field is **RO**.

Accessing AMEVTYPER0<n>_EL0

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPER0<n>_EL0 are UNDEFINED.

Note

AMCGCR_EL0.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVTYPER0<m>_EL0 ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b011:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER0_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER0_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER0_EL0[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVTYPER0_EL0[m];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVTYPER1<n>_EL0, Activity Monitors Event Type Registers 1, n = 0 - 15

The AMEVTYPER1<n>_EL0 characteristics are:

Purpose

Provides information on the events that an auxiliary activity monitor event counter AMEVCNTR1<n>_EL0 counts.

Configuration

AArch64 System register AMEVTYPER1<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER1<n>[31:0].

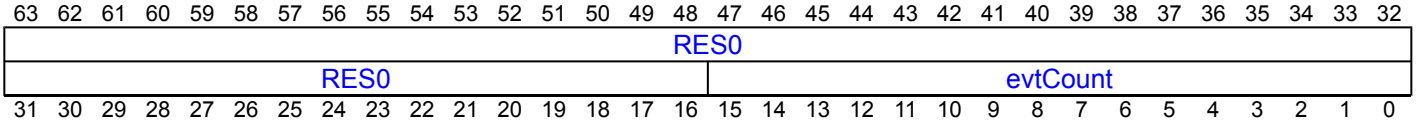
AArch64 System register AMEVTYPER1<n>_EL0 bits [31:0] are architecturally mapped to External register AMEVTYPER1<n>[31:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER1<n>_EL0 are UNDEFINED.

Attributes

AMEVTYPER1<n>_EL0 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter AMEVCNTR1<n>_EL0.

It is IMPLEMENTATION DEFINED what values are supported by each counter.

If software writes a value to this field which is not supported by the corresponding counter AMEVCNTR1<n>_EL0, then:

- It is UNPREDICTABLE which event will be counted.
- The value read back is UNKNOWN.

The event counted by AMEVCNTR1<n>_EL0 might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.

If the corresponding counter AMEVCNTR1<n>_EL0 is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVTYPER1<n>_EL0

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVTYPER1<n>_EL0 are UNDEFINED.

Note

AMCGCR_EL0.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVTYPER1<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b111:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVTYPER1<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER1_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HAFGRTR_EL2.AMEVTYPER1<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER1_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER1_EL0[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVTYPER1_EL0[m];
```

```
MSR AMEVTYPER1<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b111:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) && !boolean IMPLEMENTATION_DEFINED "AMEVCNTR1_EL0[m] is fixed" then
    AMEVTYPER1_EL0[m] = X[t, 64];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMUSERENR_EL0, Activity Monitors User Enable Register

The AMUSERENR_EL0 characteristics are:

Purpose

Global user enable register for the activity monitors. Enables or disables EL0 access to the activity monitors. AMUSERENR_EL0 is applicable to both the architected and the auxiliary counter groups.

Configuration

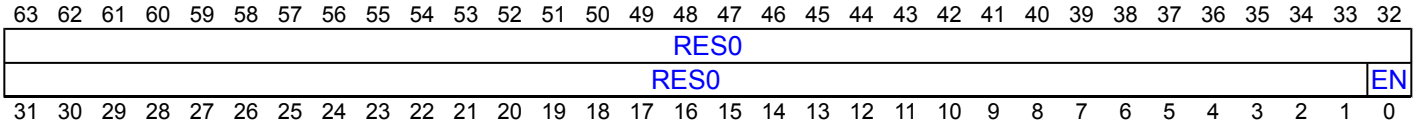
AArch64 System register AMUSERENR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMUSERENR[31:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMUSERENR_EL0 are UNDEFINED.

Attributes

AMUSERENR_EL0 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

EN, bit [0]

Traps EL0 accesses to the activity monitors registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped, reported using EC syndrome value `0x18`:
  - AMCFGR_EL0, AMCGCR_EL0, AMCNTENCLR0_EL0, AMCNTENCLR1_EL0, AMCNTENSET0_EL0, AMCNTENSET1_EL0, AMCR_EL0, AMEVCNTR0<n>_EL0, AMEVCNTR1<n>_EL0, AMEVTYPER0<n>_EL0, and AMEVTYPER1<n>_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03`, MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`:
  - AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTENSET0, AMCNTENSET1, AMCR, AMEVCNTR0<n>, AMEVCNTR1<n>, AMEVTYPER0<n>, and AMEVTYPER1<n>.

EN	Meaning
`0b0`	EL0 accesses to the activity monitors registers are trapped.
`0b1`	This control does not cause any instructions to be trapped. Software can access all activity monitor registers at EL0.

Note

- AMUSERENR_EL0 can always be read at EL0 and is not governed by this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMUSERENR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMUSERENR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMUSERENR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMUSERENR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMUSERENR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMUSERENR_EL0;
```

```
MSR AMUSERENR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMUSERENR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMUSERENR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMUSERENR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APAS, Associate PA space

The APAS characteristics are:

Purpose

Associate a location with a specified PA space, for locations that are protected by a memory-side PAS filter.

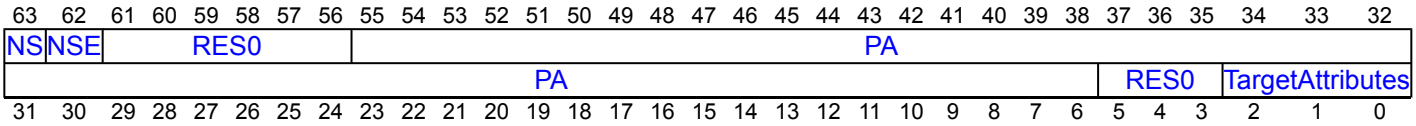
Configuration

This instruction is present only when FEAT_RME_GPC3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APAS are UNDEFINED.

Attributes

APAS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

Together with the NSE field, selects the target physical address space.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

NSE, bit [62]

Together with the NS field, selects the target physical address space.

For a description of the values derived by evaluating NS and NSE together, see .NS.

Bits [61:56]

Reserved, RES0.

PA, bits [55:6]

Bits [55:6] of the Physical Address.

Bits of Xt corresponding to physical address bits above the implemented physical address size indicated in ID_AA64MMFR0_EL1.PARange are RES0. For example, if 44 bits of PA space are implemented then Xt[55:44] are RES0.

Bits [5:3]

Reserved, RES0.

TargetAttributes, bits [2:0]

The Target Attributes field is encoded as follows:

TargetAttributes	Meaning
0b000	Default behavior applies.
0b001..0b111	IMPLEMENTATION DEFINED.

Executing APAS

This instruction is not subject to any translation, permission checks, or granule protection checks.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
APAS <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b0111	0b0000	0b000

```
if !(IsFeatureImplemented(FEAT_RME_GPC3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.APAS(X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APDAKeyHi_EL1, Pointer Authentication Key A for Data (bits[127:64])

The APDAKeyHi_EL1 characteristics are:

Purpose

Holds bits[127:64] of key A used for authentication of data pointer values.

Note

The term APDAKey_EL1 is used to describe the concatenation of APDAKeyHi_EL1: APDAKeyLo_EL1.

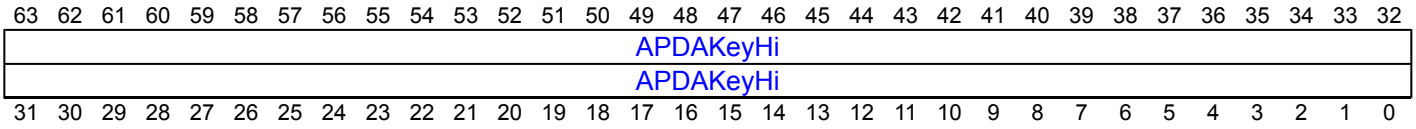
Configuration

This register is present only when FEAT_PAuth is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APDAKeyHi_EL1 are UNDEFINED.

Attributes

APDAKeyHi_EL1 is a 64-bit register.

Field descriptions



APDAKeyHi, bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APDAKeyHi_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, APDAKeyHi_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.APDAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APDAKeyHi_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APDAKeyHi_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = APDAKeyHi_EL1;
```

```
MSR APDAKeyHi_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.APDAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDAKeyHi_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDAKeyHi_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    APDAKeyHi_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APDAKeyLo_EL1, Pointer Authentication Key A for Data (bits[63:0])

The APDAKeyLo_EL1 characteristics are:

Purpose

Holds bits[63:0] of key A used for authentication of data pointer values.

Note

The term APDAKey_EL1 is used to describe the concatenation of APDAKeyHi_EL1: APDAKeyLo_EL1.

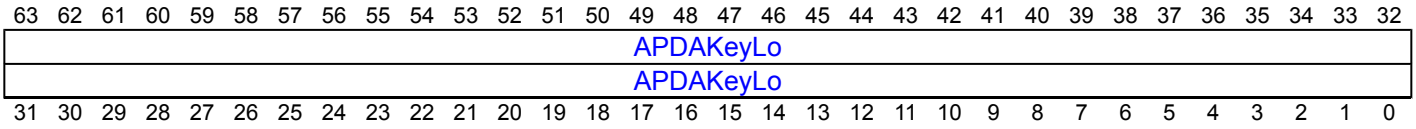
Configuration

This register is present only when FEAT_PAuth is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APDAKeyLo_EL1 are UNDEFINED.

Attributes

APDAKeyLo_EL1 is a 64-bit register.

Field descriptions



APDAKeyLo, bits [63:0]

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APDAKeyLo_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, APDAKeyLo_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0010	0b000

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.APDAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APDAKeyLo_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APDAKeyLo_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = APDAKeyLo_EL1;
```

```
MSR APDAKeyLo_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0010	0b000

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.APDAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDAKeyLo_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDAKeyLo_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    APDAKeyLo_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APDBKeyHi_EL1, Pointer Authentication Key B for Data (bits[127:64])

The APDBKeyHi_EL1 characteristics are:

Purpose

Holds bits[127:64] of key B used for authentication of data pointer values.

Note

The term APDBKey_EL1 is used to describe the concatenation of APDBKeyHi_EL1: APDBKeyLo_EL1.

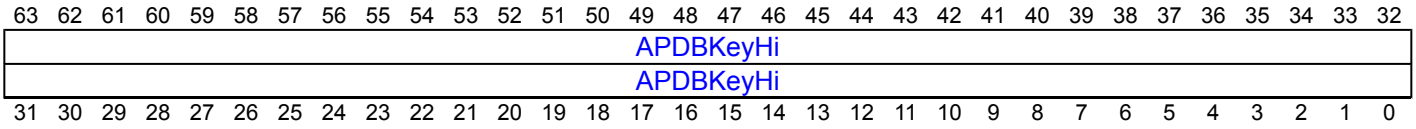
Configuration

This register is present only when FEAT_PAuth is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APDBKeyHi_EL1 are UNDEFINED.

Attributes

APDBKeyHi_EL1 is a 64-bit register.

Field descriptions



APDBKeyHi, bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APDBKeyHi_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, APDBKeyHi_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.APDBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APDBKeyHi_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APDBKeyHi_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = APDBKeyHi_EL1;
```

MSR APDBKeyHi_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.APDBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDBKeyHi_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDBKeyHi_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    APDBKeyHi_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APDBKeyLo_EL1, Pointer Authentication Key B for Data (bits[63:0])

The APDBKeyLo_EL1 characteristics are:

Purpose

Holds bits[63:0] of key B used for authentication of data pointer values.

Note

The term APDBKey_EL1 is used to describe the concatenation of APDBKeyHi_EL1: APDBKeyLo_EL1.

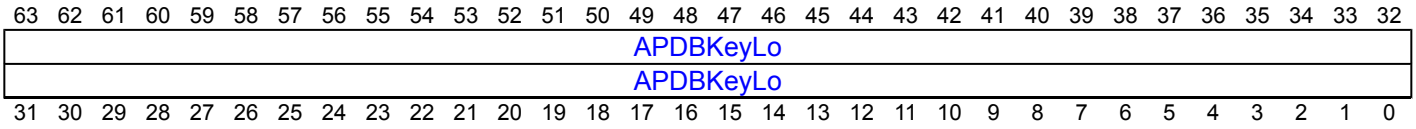
Configuration

This register is present only when FEAT_PAuth is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APDBKeyLo_EL1 are UNDEFINED.

Attributes

APDBKeyLo_EL1 is a 64-bit register.

Field descriptions



APDBKeyLo, bits [63:0]

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APDBKeyLo_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, APDBKeyLo_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.APDBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APDBKeyLo_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APDBKeyLo_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = APDBKeyLo_EL1;
```

```
MSR APDBKeyLo_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.APDBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDBKeyLo_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDBKeyLo_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    APDBKeyLo_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APGAKeyHi_EL1, Pointer Authentication Key A for Code (bits[127:64])

The APGAKeyHi_EL1 characteristics are:

Purpose

Holds bits[127:64] of key used for generic pointer authentication code.

Note

The term APGAKey_EL1 is used to describe the concatenation of APGAKeyHi_EL1: APGAKeyLo_EL1.

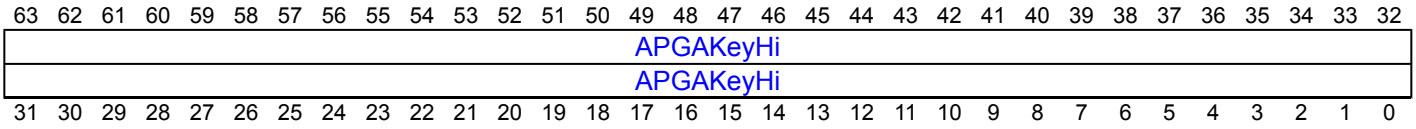
Configuration

This register is present only when FEAT_PAuth is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APGAKeyHi_EL1 are UNDEFINED.

Attributes

APGAKeyHi_EL1 is a 64-bit register.

Field descriptions



APGAKeyHi, bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APGAKeyHi_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, APGAKeyHi_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.APGAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APGAKeyHi_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APGAKeyHi_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = APGAKeyHi_EL1;
```

```
MSR APGAKeyHi_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.APGAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APGAKeyHi_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APGAKeyHi_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    APGAKeyHi_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APGAKeyLo_EL1, Pointer Authentication Key A for Code (bits[63:0])

The APGAKeyLo_EL1 characteristics are:

Purpose

Holds bits[63:0] of key used for generic pointer authentication code.

Note

The term APGAKey_EL1 is used to describe the concatenation of APGAKeyHi_EL1: APGAKeyLo_EL1.

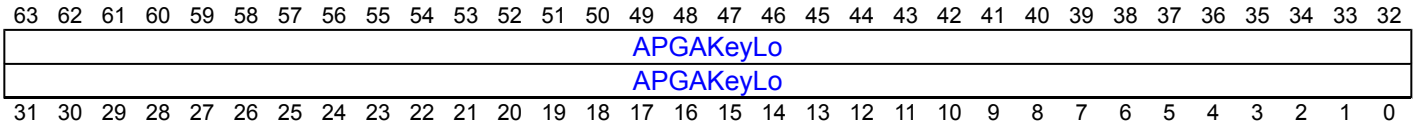
Configuration

This register is present only when FEAT_PAuth is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APGAKeyLo_EL1 are UNDEFINED.

Attributes

APGAKeyLo_EL1 is a 64-bit register.

Field descriptions



APGAKeyLo, bits [63:0]

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APGAKeyLo_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, APGAKeyLo_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.APGAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APGAKeyLo_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APGAKeyLo_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = APGAKeyLo_EL1;
```

```
MSR APGAKeyLo_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.APGAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APGAKeyLo_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APGAKeyLo_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    APGAKeyLo_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APIAKeyHi_EL1, Pointer Authentication Key A for Instruction (bits[127:64])

The APIAKeyHi_EL1 characteristics are:

Purpose

Holds bits[127:64] of key A used for authentication of instruction pointer values.

Note

The term APIAKey_EL1 is used to describe the concatenation of APIAKeyHi_EL1: APIAKeyLo_EL1.

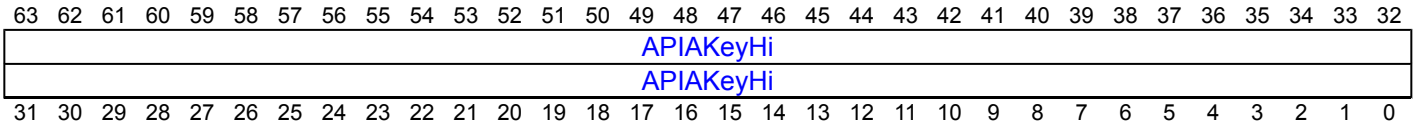
Configuration

This register is present only when FEAT_PAuth is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APIAKeyHi_EL1 are UNDEFINED.

Attributes

APIAKeyHi_EL1 is a 64-bit register.

Field descriptions



APIAKeyHi, bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APIAKeyHi_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, APIAKeyHi_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.APIAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APIAKeyHi_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APIAKeyHi_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = APIAKeyHi_EL1;
```

```
MSR APIAKeyHi_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.APIAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIAKeyHi_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIAKeyHi_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    APIAKeyHi_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APIAKeyLo_EL1, Pointer Authentication Key A for Instruction (bits[63:0])

The APIAKeyLo_EL1 characteristics are:

Purpose

Holds bits[63:0] of key A used for authentication of instruction pointer values.

Note

The term APIAKey_EL1 is used to describe the concatenation of APIAKeyHi_EL1: APIAKeyLo_EL1.

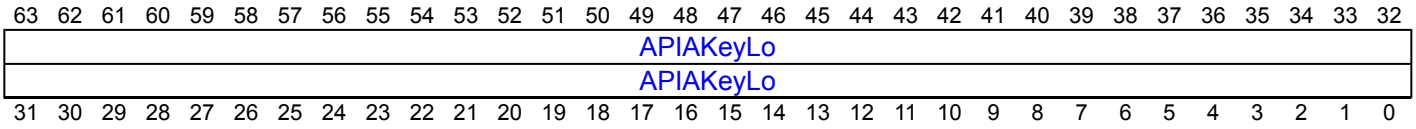
Configuration

This register is present only when FEAT_PAuth is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APIAKeyLo_EL1 are UNDEFINED.

Attributes

APIAKeyLo_EL1 is a 64-bit register.

Field descriptions



APIAKeyLo, bits [63:0]

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APIAKeyLo_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, APIAKeyLo_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.APIAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APIAKeyLo_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APIAKeyLo_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = APIAKeyLo_EL1;
```

```
MSR APIAKeyLo_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.APIAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIAKeyLo_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIAKeyLo_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    APIAKeyLo_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APIBKeyHi_EL1, Pointer Authentication Key B for Instruction (bits[127:64])

The APIBKeyHi_EL1 characteristics are:

Purpose

Holds bits[127:64] of key B used for authentication of instruction pointer values.

Note

The term APIBKey_EL1 is used to describe the concatenation of APIBKeyHi_EL1: APIBKeyLo_EL1.

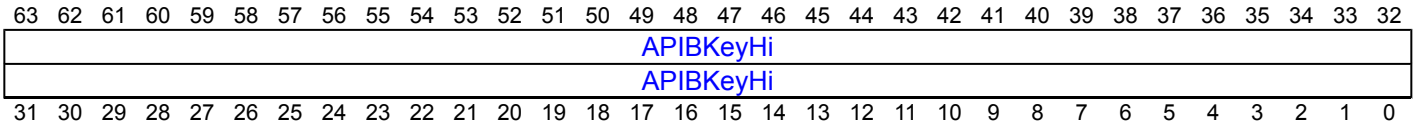
Configuration

This register is present only when FEAT_PAuth is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APIBKeyHi_EL1 are UNDEFINED.

Attributes

APIBKeyHi_EL1 is a 64-bit register.

Field descriptions



APIBKeyHi, bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APIBKeyHi_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, APIBKeyHi_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0001	0b011

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APIBKeyHi_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APIBKeyHi_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = APIBKeyHi_EL1;
```

```
MSR APIBKeyHi_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0001	0b011

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIBKeyHi_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIBKeyHi_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    APIBKeyHi_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APIBKeyLo_EL1, Pointer Authentication Key B for Instruction (bits[63:0])

The APIBKeyLo_EL1 characteristics are:

Purpose

Holds bits[63:0] of key B used for authentication of instruction pointer values.

Note

The term APIBKey_EL1 is used to describe the concatenation of APIBKeyHi_EL1: APIBKeyLo_EL1.

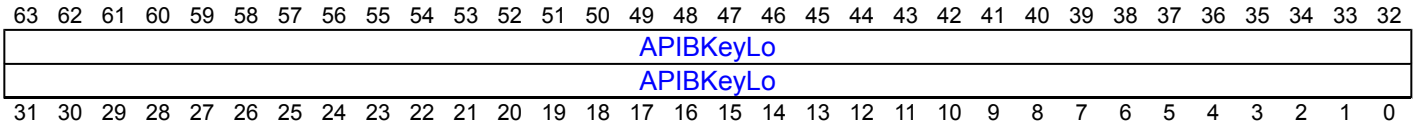
Configuration

This register is present only when FEAT_PAuth is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to APIBKeyLo_EL1 are UNDEFINED.

Attributes

APIBKeyLo_EL1 is a 64-bit register.

Field descriptions



APIBKeyLo, bits [63:0]

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APIBKeyLo_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, APIBKeyLo_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APIBKeyLo_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = APIBKeyLo_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = APIBKeyLo_EL1;
```

```
MSR APIBKeyLo_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_PAuth) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIBKeyLo_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIBKeyLo_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    APIBKeyLo_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S12E0R, Address Translate Stages 1 and 2 EL0 Read

The AT S12E0R characteristics are:

Purpose

Performs stage 1 and 2 address translation, with permissions as if reading from the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

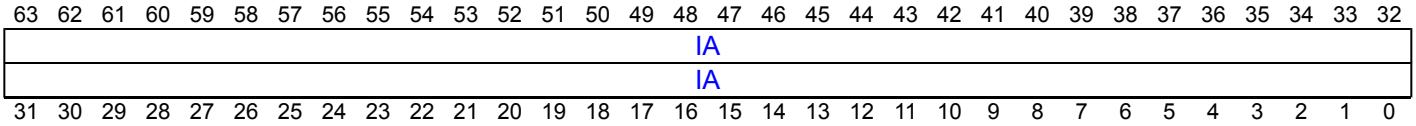
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S12E0R are UNDEFINED.

Attributes

AT S12E0R is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S12E0R

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S12E0R, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b0111	0b1000	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) || HCR_EL2.<DC,VM> == '00' then
        AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Read);
    else
        AArch64.AT(X[t, 64], TranslationStage_12, EL0, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Read);
    elsif EL2Enabled() && (ELIsInHost(EL0) || HCR_EL2.<DC,VM> == '00') then
        AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Read);
    else
        AArch64.AT(X[t, 64], TranslationStage_12, EL0, ATAccess_Read);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S12E0W, Address Translate Stages 1 and 2 EL0 Write

The AT S12E0W characteristics are:

Purpose

Performs stage 1 and 2 address translation, with permissions as if writing to the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

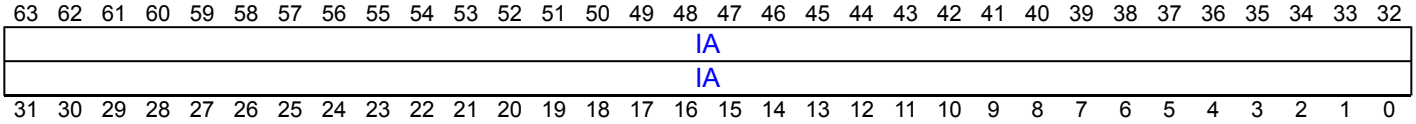
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S12E0W are UNDEFINED.

Attributes

AT S12E0W is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S12E0W

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S12E0W, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b0111	0b1000	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) || HCR_EL2.<DC,VM> == '00' then
        AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Write);
    else
        AArch64.AT(X[t, 64], TranslationStage_12, EL0, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Write);
    elsif EL2Enabled() && (ELIsInHost(EL0) || HCR_EL2.<DC,VM> == '00') then
        AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Write);
    else
        AArch64.AT(X[t, 64], TranslationStage_12, EL0, ATAccess_Write);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S12E1R, Address Translate Stages 1 and 2 EL1 Read

The AT S12E1R characteristics are:

Purpose

Performs stage 1 and 2 address translation, with permissions as if reading from the given virtual address from EL1, or from EL2 if the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

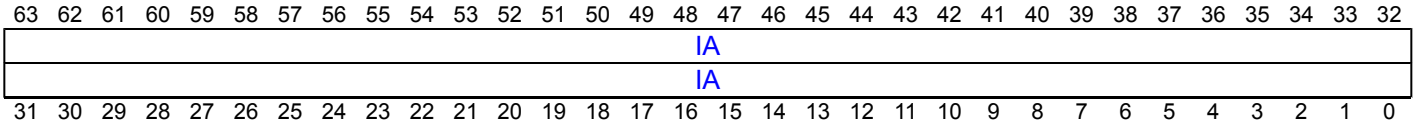
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S12E1R are UNDEFINED.

Attributes

AT S12E1R is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S12E1R

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S12E1R, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b0111	0b1000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) || HCR_EL2.<DC,VM> == '00' then
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Read);
    else
        AArch64.AT(X[t, 64], TranslationStage_12, EL1, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Read);
    elsif EL2Enabled() && (ELIsInHost(EL0) || HCR_EL2.<DC,VM> == '00') then
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Read);
    else
        AArch64.AT(X[t, 64], TranslationStage_12, EL1, ATAccess_Read);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S12E1W, Address Translate Stages 1 and 2 EL1 Write

The AT S12E1W characteristics are:

Purpose

Performs stage 1 and 2 address translation, with permissions as if writing to the given virtual address from EL1, or from EL2 if the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

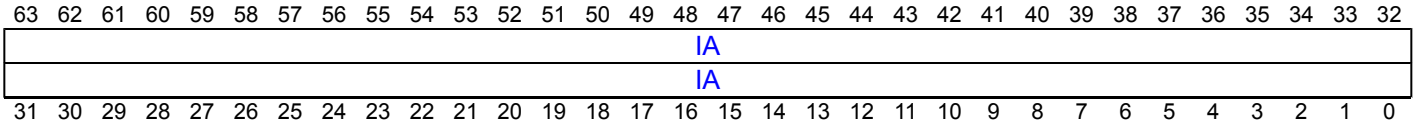
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S12E1W are UNDEFINED.

Attributes

AT S12E1W is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S12E1W

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S12E1W, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b0111	0b1000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) || HCR_EL2.<DC,VM> == '00' then
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Write);
    else
        AArch64.AT(X[t, 64], TranslationStage_12, EL1, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Write);
    elsif EL2Enabled() && (ELIsInHost(EL0) || HCR_EL2.<DC,VM> == '00') then
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Write);
    else
        AArch64.AT(X[t, 64], TranslationStage_12, EL1, ATAccess_Write);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E0R, Address Translate Stage 1 EL0 Read

The AT S1E0R characteristics are:

Purpose

Performs stage 1 address translation, with permissions as if reading from the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

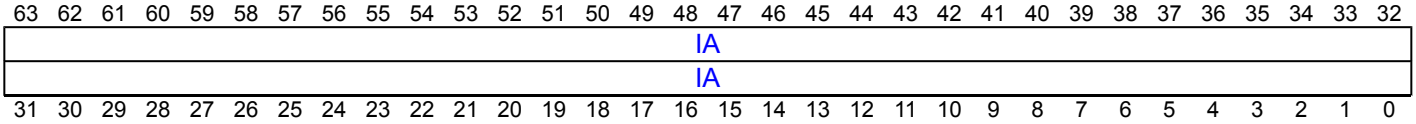
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E0R are UNDEFINED.

Attributes

AT S1E0R is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S1E0R

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E0R, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.ATS1E0R == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Read);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Read);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E0W, Address Translate Stage 1 EL0 Write

The AT S1E0W characteristics are:

Purpose

Performs stage 1 address translation, with permissions as if writing to the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

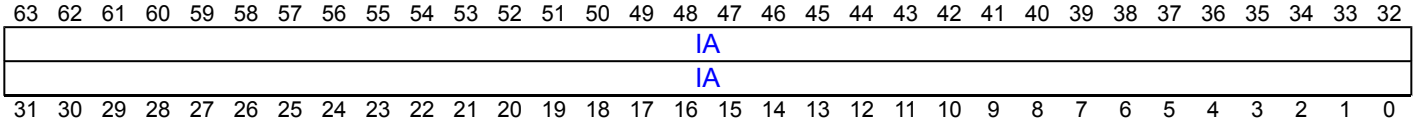
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E0W are UNDEFINED.

Attributes

AT S1E0W is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S1E0W

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E0W, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1000	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.ATS1E0W == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Write);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL0, ATAccess_Write);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E1A, Address Translate Stage 1 EL1 Without Permission checks

The AT S1E1A characteristics are:

Purpose

Performs a stage 1 address translation, while ignoring the permission checks using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

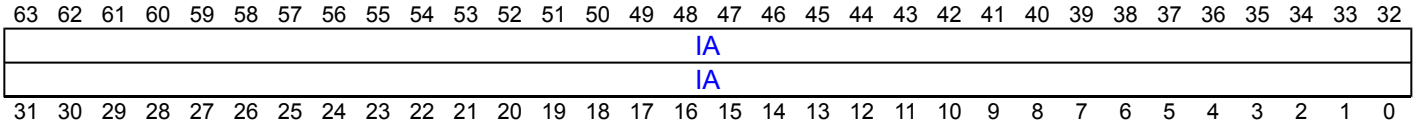
Configuration

This instruction is present only when FEAT_ATS1A is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E1A are UNDEFINED.

Attributes

AT S1E1A is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

Executing AT S1E1A

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E1A, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1001	0b010

```
if !(IsFeatureImplemented(FEAT_ATS1A) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.ATS1E1A == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Any);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Any);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Any);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E1R, Address Translate Stage 1 EL1 Read

The AT S1E1R characteristics are:

Purpose

Performs stage 1 address translation, with permissions as if reading from the given virtual address from EL1, or from EL2 if the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

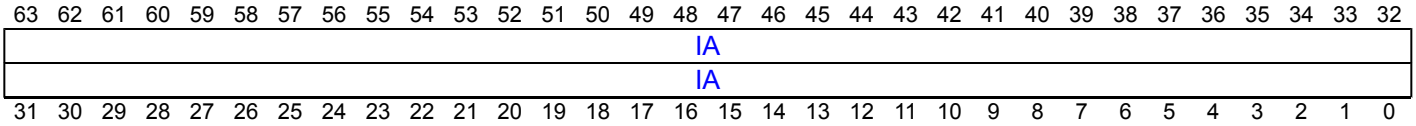
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E1R are UNDEFINED.

Attributes

AT S1E1R is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S1E1R

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E1R, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.ATS1E1R == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Read);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Read);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E1RP, Address Translate Stage 1 EL1 Read PAN

The AT S1E1RP characteristics are:

Purpose

Performs a stage 1 address translation, where the value of PSTATE.PAN determines if a read from a location will generate a Permission fault for a privileged access, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

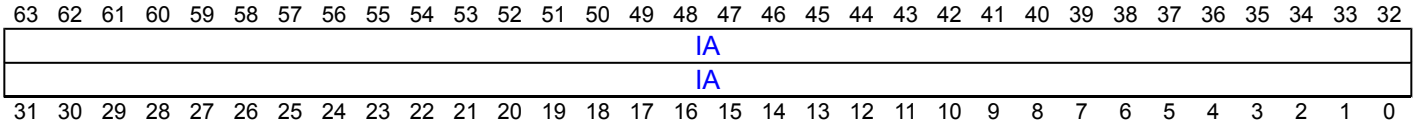
Configuration

This instruction is present only when FEAT_PAN2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E1RP are UNDEFINED.

Attributes

AT S1E1RP is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S1E1RP

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E1RP, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1001	0b000

```
if !(IsFeatureImplemented(FEAT_PAN2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.ATS1E1RP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_ReadPAN);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_ReadPAN);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_ReadPAN);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E1W, Address Translate Stage 1 EL1 Write

The AT S1E1W characteristics are:

Purpose

Performs stage 1 address translation, with permissions as if writing to the given virtual address from EL1, or from EL2 if the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

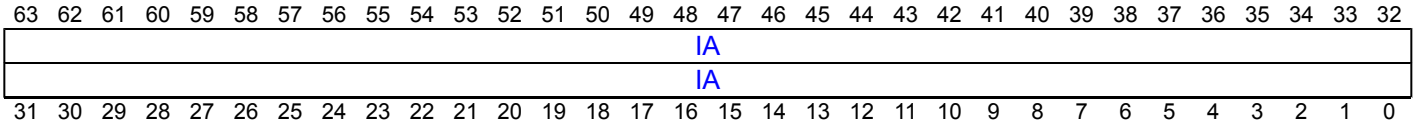
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E1W are UNDEFINED.

Attributes

AT S1E1W is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S1E1W

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E1W, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.ATS1E1W == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Write);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_Write);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E1WP, Address Translate Stage 1 EL1 Write PAN

The AT S1E1WP characteristics are:

Purpose

Performs a stage 1 address translation, where the value of PSTATE.PAN determines if a write to a location will generate a Permission fault for a privileged access, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current Effective value of SCR_EL3.{NSE, NS}:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

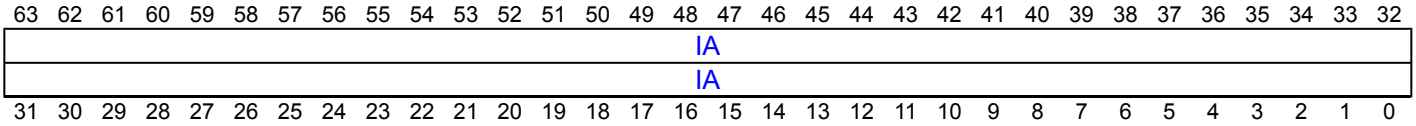
Configuration

This instruction is present only when FEAT_PAN2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E1WP are UNDEFINED.

Attributes

AT S1E1WP is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S1E1WP

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E1WP, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1001	0b001

```
if !(IsFeatureImplemented(FEAT_PAN2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.ATS1E1WP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_WritePAN);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_WritePAN);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL1, ATAccess_WritePAN);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E2A, Address Translate Stage 1 EL2 Without Permission checks

The AT S1E2A characteristics are:

Purpose

Performs stage 1 address translation as defined for EL2, while ignoring permissions checks from the given virtual address.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

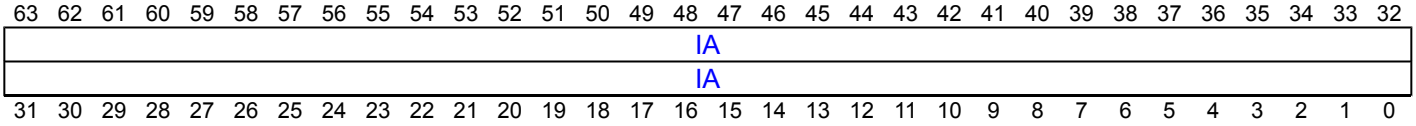
Configuration

This instruction is present only when FEAT_ATS1A is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E2A are UNDEFINED.

Attributes

AT S1E2A is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

Executing AT S1E2A

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E2A, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b0111	0b1001	0b010

```
if !(IsFeatureImplemented(FEAT_ATS1A) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL2, ATAccess_Any);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.AT(X[t, 64], TranslationStage_1, EL2, ATAccess_Any);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E2R, Address Translate Stage 1 EL2 Read

The AT S1E2R characteristics are:

Purpose

Performs stage 1 address translation as defined for EL2, with permissions as if reading from the given virtual address.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

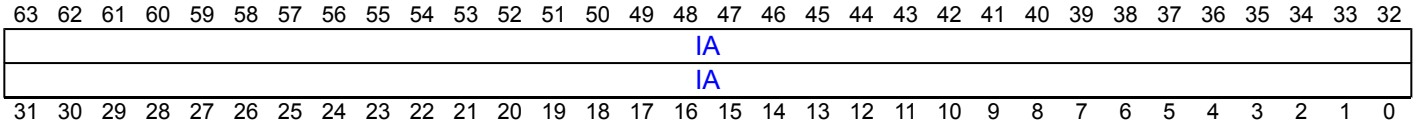
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E2R are UNDEFINED.

Attributes

AT S1E2R is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S1E2R

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E2R, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b0111	0b1000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL2, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.AT(X[t, 64], TranslationStage_1, EL2, ATAccess_Read);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E2W, Address Translate Stage 1 EL2 Write

The AT S1E2W characteristics are:

Purpose

Performs stage 1 address translation as defined for EL2, with permissions as if writing to the given virtual address.

When FEAT_RME is implemented, if the Effective value of SCR_EL3.{NSE, NS} is a reserved value, this instruction is UNDEFINED at EL3.

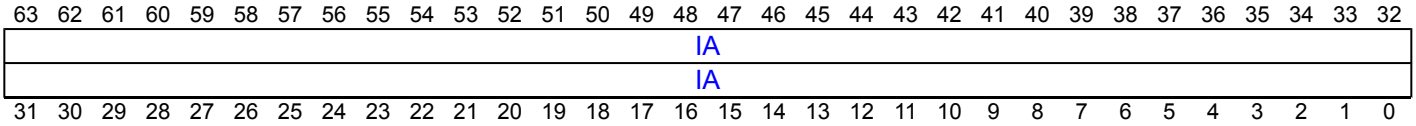
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E2W are UNDEFINED.

Attributes

AT S1E2W is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S1E2W

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E2W, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b0111	0b1000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL2, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.AT(X[t, 64], TranslationStage_1, EL2, ATAccess_Write);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E3A, Address Translate Stage 1 EL3 Without Permission checks

The AT S1E3A characteristics are:

Purpose

Performs stage 1 address translation as defined for EL3, while ignoring permissions checks from the given virtual address.

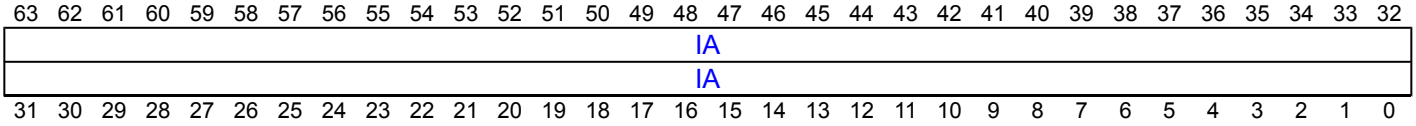
Configuration

This instruction is present only when FEAT_ATS1A is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E3A are UNDEFINED.

Attributes

AT S1E3A is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

Executing AT S1E3A

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E3A, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b0111	0b1001	0b010

```
if !(IsFeatureImplemented(FEAT_ATS1A) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL3, ATAccess_Any);
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E3R, Address Translate Stage 1 EL3 Read

The AT S1E3R characteristics are:

Purpose

Performs stage 1 address translation as defined for EL3, with permissions as if reading from the given virtual address.

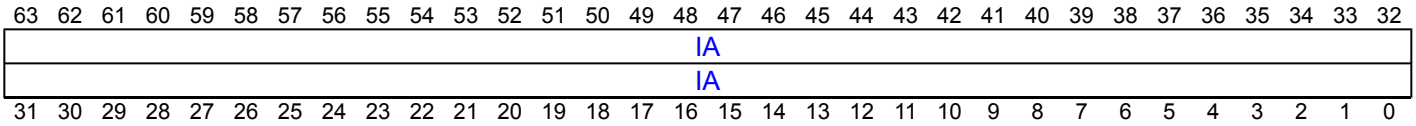
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E3R are UNDEFINED.

Attributes

AT S1E3R is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S1E3R

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E3R, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b0111	0b1000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL3, ATAccess_Read);
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E3W, Address Translate Stage 1 EL3 Write

The AT S1E3W characteristics are:

Purpose

Performs stage 1 address translation as defined for EL3, with permissions as if writing to the given virtual address.

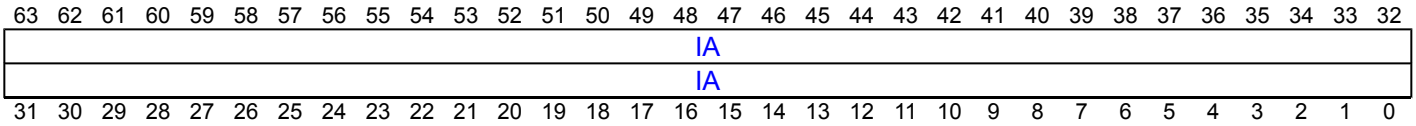
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to AT S1E3W are UNDEFINED.

Attributes

AT S1E3W is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing AT S1E3W

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
AT S1E3W, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b0111	0b1000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t, 64], TranslationStage_1, EL3, ATAccess_Write);
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRB IALL, Invalidate the Branch Record Buffer

The BRB IALL characteristics are:

Purpose

Invalidates all Branch records in the Branch Record Buffer.

Configuration

This instruction is present only when FEAT_BRBE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to BRB IALL are UNDEFINED.

Attributes

BRB IALL is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing BRB IALL

Rt should be encoded as `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
BRB IALL
```

op0	op1	CRn	CRm	op2
0b01	0b001	0b0111	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_BRBE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.nBRBIALL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRB_IALL();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRB_IALL();
elsif PSTATE.EL == EL3 then
    BRB_IALL();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRB INJ, Branch Record Injection into the Branch Record Buffer

The BRB INJ characteristics are:

Purpose

Injects the Branch Record held in BRBINFINJ_EL1, BRBSRCINJ_EL1, and BRBTGTINJ_EL1 into the Branch Record Buffer.

Configuration

This instruction is present only when FEAT_BRBE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to BRB INJ are UNDEFINED.

Attributes

BRB INJ is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing BRB INJ

Rt should be encoded as `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

`BRB INJ`

op0	op1	CRn	CRm	op2
0b01	0b001	0b0111	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_BRBE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.nBRBINJ == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRB_INJ();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRB_INJ();
elsif PSTATE.EL == EL3 then
    BRB_INJ();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBCR_EL1, Branch Record Buffer Control Register (EL1)

The BRBCR_EL1 characteristics are:

Purpose

Controls the Branch Record Buffer.

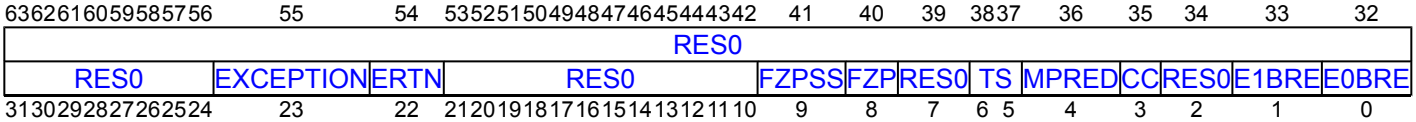
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBCR_EL1 are UNDEFINED.

Attributes

BRBCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

EXCEPTION, bit [23]

Enable the recording of entry to EL1 via an exception.

EXCEPTION	Meaning
0b0	Disable the recording of Branch records for exceptions when taken to EL1.
0b1	Enable the recording of Branch records for exceptions when taken to EL1.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

ERTN, bit [22]

Allow the recording Branch records for exception return instructions from EL1.

ERTN	Meaning
0b0	Disable the recording Branch records for exception return instructions from EL1.
0b1	Enable the recording Branch records for exception return instructions from EL1.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bits [21:10]

Reserved, RES0.

FZPSS, bit [9]

When FEAT_PMUv3_SS is implemented:

Freeze BRBE on PMU Snapshot.

FZPSS	Meaning
0b0	Branch recording is not affected by this control.
0b1	If either EL2 is not implemented or BRBCR_EL2.FZPSS is 1, then a BRBE freeze event occurs when a successful Capture event occurs.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FZP, bit [8]

When FEAT_PMUv3 is implemented:

Freeze BRBE on PMU overflow.

FZP	Meaning
0b0	Branch recording is not affected by this control.
0b1	A BRBE freeze event occurs when the PE is in a Non-prohibited region, BRBFCR_EL1.PAUSED is 0, and any of the following applies: • For any event counter PMEVCNTR<m>_EL0 in the first range, PMOVSCLR_EL0[m] is 1, and either FEAT_SEBEP is not implemented or PMEVTYPER<m>_EL0.SYNC is 0. • FEAT_PMUv3_ICNTR is implemented, PMOVSCLR_EL0.F0 is 1, and either FEAT_SEBEP is not implemented or PMICFILTR_EL0.SYNC is 0. For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control.

TS	Meaning	Applies when
0b01	Virtual timestamp. The BRBE recorded timestamp is the physical counter value, minus the value of CNTVOFF_EL2.	
0b10	Guest physical timestamp. The BRBE recorded timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • EL3 is implemented and SCR_EL3.ECVEn == 0. • EL2 is implemented and CNTHCTL_EL2.ECV == 0. • FEAT_ECV_POFF is not implemented.	When FEAT_ECV is implemented
0b11	Physical timestamp. The BRBE recorded timestamp is the physical counter value.	

All other values are reserved.

This field is ignored by the PE when EL2 is implemented and BRBCR_EL2.TS != 0b00.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

MPRED, bit [4]

Mask the recording of mispredicts.

MPRED	Meaning
0b0	Disable the recording of mispredict information.
0b1	Allow the recording of mispredict information.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

CC, bit [3]

Enable the recording of cycle count information.

CC	Meaning
0b0	Disable the recording of cycle count information.
0b1	Allow the recording of cycle count information.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E1BRE, bit [1]

EL1 Branch recording enable.

E1BRE	Meaning
0b0	Branch recording prohibited at EL1.
0b1	Branch recording enabled at EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0BRE, bit [0]

EL0 Branch recording enable.

E0BRE	Meaning
0b0	Branch recording prohibited at EL0.
0b1	Branch recording enabled at EL0.

This field is ignored by the PE when all of the following are true:

- HCR_EL2.TGE == 1.
- EL2 is implemented and enabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing BRBCR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name BRBCR_EL1 or BRBCR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8E0];
    else
        X[t, 64] = BRBCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = BRBCR_EL2;
    else
        X[t, 64] = BRBCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBCR_EL1;
```

When FEAT_VHE is implemented

```
MRS <Xt>, BRBCR_EL12
```

op0	op1	CRn	CRm	op2
0b10	0b101	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x8E0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS ==
'1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = BRBCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = BRBCR_EL1;
    else
        UNDEFINED;
```

MSR BRBCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8E0] = X[t, 64];
    else
        BRBCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        BRBCR_EL2 = X[t, 64];
    else
        BRBCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MSR BRBCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b101	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x8E0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS ==
'1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            BRBCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        BRBCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBCR_EL2, Branch Record Buffer Control Register (EL2)

The BRBCR_EL2 characteristics are:

Purpose

Controls the Branch Record Buffer.

Configuration

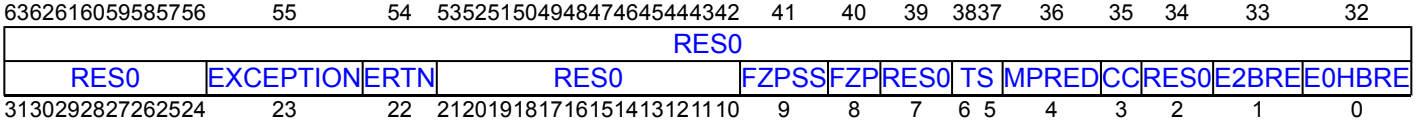
This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBCR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

BRBCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

EXCEPTION, bit [23]

Enable the recording of entry to EL2 via an exception.

EXCEPTION	Meaning
0b0	Disable the recording of Branch records for exceptions when taken to EL2.
0b1	Enable the recording of Branch records for exceptions when taken to EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

ERTN, bit [22]

Allow the recording Branch records for exception return instructions from EL2.

ERTN	Meaning
0b0	Disable the recording Branch records for exception return instructions from EL2.
0b1	Enable the recording Branch records for exception return instructions from EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bits [21:10]

Reserved, RES0.

FZPSS, bit [9]

When FEAT_PMUv3_SS is implemented:

Freeze BRBE on PMU Snapshot.

FZPSS	Meaning
0b0	Branch recording is not affected by this control.
0b1	If BRBCR_EL1.FZPSS is 1, then a BRBE freeze event occurs when a PMU snapshot occurs.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FZP, bit [8]

When FEAT_PMUv3 is implemented:

Freeze BRBE on PMU overflow.

FZP	Meaning
0b0	Branch recording is not affected by this control.
0b1	A BRBE freeze event occurs when the PE is in a Non-prohibited region, BRBFCR_EL1.PAUSED is 0, and all the following are true for any event counter PMEVCNTR<m>_EL0 in the second range: • PMOVSCLR_EL0[m] is 1. • Either FEAT_SEBEP is not implemented or PMEVTYPER<m>_EL0.SYNC is 0. For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control.

TS	Meaning	Applies when
0b00	Timestamp controlled by BRBCR_EL1.TS.	
0b01	Virtual timestamp. The BRBE recorded timestamp is the physical counter value, minus the value of CNTVOFF_EL2.	
0b10	Guest physical timestamp. The BRBE recorded timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • EL3 is implemented and SCR_EL3.ECVEn == 0. • EL2 is implemented and CNTHCTL_EL2.ECV == 0. • FEAT_ECV_POFF is not implemented.	When FEAT_ECV is implemented
0b11	Physical timestamp. The BRBE recorded timestamp is the physical counter value.	

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

MPRED, bit [4]

Mask the recording of mispredicts.

MPRED	Meaning
0b0	Disable the recording of mispredict information.
0b1	Allow the recording of mispredict information.

If EL2 is not implemented, then the Effective value of this field is 1, other than for a direct read of the register.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

CC, bit [3]

Enable the recording of cycle count information.

CC	Meaning
0b0	Disable the recording of cycle count information.
0b1	Allow the recording of cycle count information.

If EL2 is not implemented, then the Effective value of this field is 1, other than for a direct read of the register.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E2BRE, bit [1]

EL2 Branch recording enable.

E2BRE	Meaning
0b0	Branch recording prohibited at EL2.
0b1	Branch recording enabled at EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0HBRE, bit [0]

EL0 Branch recording enable.

E0HBRE	Meaning
0b0	Branch recording prohibited at EL0 when HCR_EL2.TGE == 1.
0b1	Branch recording enabled at EL0 when HCR_EL2.TGE == 1.

This field is ignored by the PE when any of the following are true:

- HCR_EL2.TGE == 0.
- EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing BRBCR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name BRBCR_EL2 or BRBCR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8E0];
    else
        X[t, 64] = BRBCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = BRBCR_EL2;
    else
        X[t, 64] = BRBCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBCR_EL1;
```

MRS <Xt>, BRBCR_EL2

op0	op1	CRn	CRm	op2
0b10	0b100	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBCR_EL2;
```

MSR BRBCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8E0] = X[t, 64];
    else
        BRBCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        BRBCR_EL2 = X[t, 64];
    else
        BRBCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBCR_EL1 = X[t, 64];
```

```
MSR BRBCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b100	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBCR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBFCR_EL1, Branch Record Buffer Function Control Register

The BRBFCR_EL1 characteristics are:

Purpose

Functional controls for the Branch Record Buffer.

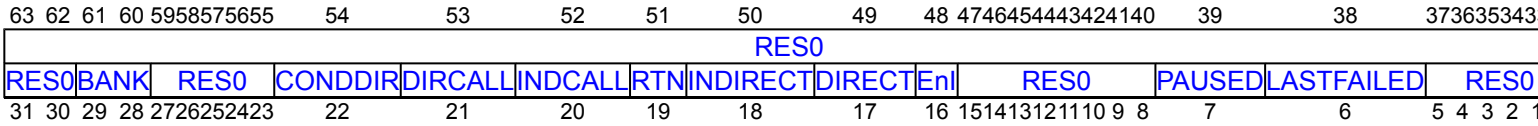
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBFCR_EL1 are UNDEFINED.

Attributes

BRBFCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:30]

Reserved, RES0.

BANK, bits [29:28]

Branch record buffer bank access control.

BANK	Meaning
0b00	Select branch records 0 to 31.
0b01	Select branch records 32 to 63.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:23]

Reserved, RES0.

CONDDIR, bit [22]

Match on conditional direct branch instructions.

CONDDIR	Meaning
0b0	Do not match on conditional direct branch instructions.
0b1	Match on conditional direct branch instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

DIRCALL, bit [21]

Match on direct branch with link instructions.

DIRCALL	Meaning
0b0	Do not match on direct branch with link instructions.
0b1	Match on direct branch with link instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

INDCALL, bit [20]

Match on indirect branch with link instructions.

INDCALL	Meaning
0b0	Do not match on indirect branch with link instructions.
0b1	Match on indirect branch with link instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

RTN, bit [19]

Match on function return instructions.

RTN	Meaning
0b0	Do not match on function return instructions.
0b1	Match on function return instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

INDIRECT, bit [18]

Match on indirect branch instructions.

INDIRECT	Meaning
0b0	Do not match on indirect branch instructions.
0b1	Match on indirect branch instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

DIRECT, bit [17]

Match on unconditional direct branch instructions.

DIRECT	Meaning
0b0	Do not match on unconditional direct branch instructions.
0b1	Match on unconditional direct branch instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

EnI, bit [16]

Include or exclude matches.

EnI	Meaning
0b0	Include records for matches, and exclude records for non-matches.
0b1	Exclude records for matches, and include records for non-matches.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bits [15:8]

Reserved, RES0.

PAUSED, bit [7]

Branch recording Paused status.

PAUSED	Meaning
0b0	Branch recording is not Paused.
0b1	Branch recording is Paused.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

LASTFAILED, bit [6]

When FEAT_TME is implemented:

Indicates transaction failure or cancellation.

LASTFAILED	Meaning
0b0	Indicates that no transactions in a non-prohibited region have failed or been canceled since the last Branch record was generated.
0b1	Indicates that at least one transaction in a non-prohibited region has failed or been canceled since the last Branch record was generated.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [5:0]

Reserved, RES0.

Accessing BRBFCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBFCR_EL1

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBFCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBFCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBFCR_EL1;
```

MSR BRBFCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBFCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBIDR0_EL1, Branch Record Buffer ID0 Register

The BRBIDR0_EL1 characteristics are:

Purpose

Indicates the features of the branch buffer unit.

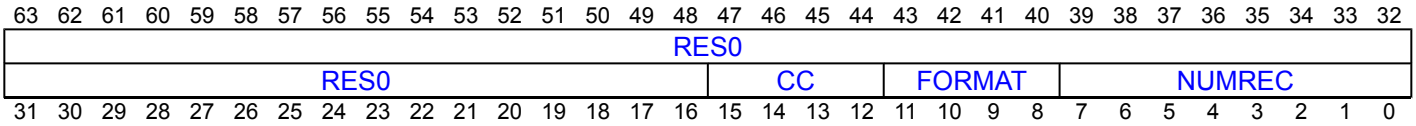
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBIDR0_EL1 are UNDEFINED.

Attributes

BRBIDR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

CC, bits [15:12]

Cycle counter support. Defined values are:

CC	Meaning
0b0101	20-bit cycle counter implemented.

All other values are reserved.

Access to this field is **RO**.

FORMAT, bits [11:8]

Data format of records of the Branch record buffer. Defined values are:

FORMAT	Meaning
0b0000	Format 0.

All other values are reserved.

Access to this field is **RO**.

NUMREC, bits [7:0]

Number of records supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMREC	Meaning
0x08	8 branch records implemented.
0x10	16 branch records implemented.
0x20	32 branch records implemented.
0x40	64 branch records implemented.

All other values are reserved.

Access to this field is **RO**.

Accessing BRBIDR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBIDR0_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBIDR == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBIDR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBIDR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBIDR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBINFINJ_EL1, Branch Record Buffer Information Injection Register

The BRBINFINJ_EL1 characteristics are:

Purpose

The information of a Branch record for injection.

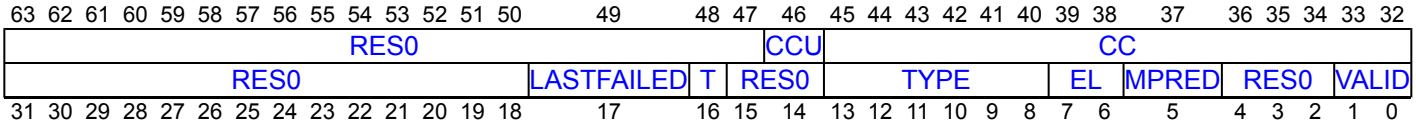
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBINFINJ_EL1 are UNDEFINED.

Attributes

BRBINFINJ_EL1 is a 64-bit register.

Field descriptions



Bits [63:47]

Reserved, RES0.

CCU, bit [46]

The number of PE clock cycles since the last Branch record entry is UNKNOWN.

CCU	Meaning
0b0	Indicates that the number of PE clock cycles since the last Branch record is indicated by BRBINFINJ_EL1.CC.
0b1	Indicates that the number of PE clock cycles since the last Branch record is UNKNOWN.

The value in this field is only valid when BRBINFINJ_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINFINJ_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

CC, bits [45:32]

The number of PE clock cycles since the last Branch record entry.

The format of this field uses a mantissa and exponent to express the cycle count value, as follows:

- CC bits[7:0] indicate the mantissa M.
- CC bits[13:8] indicate the exponent E.

The cycle count is expressed using the following function:

if IsZero(E) then UInt(M) else UInt('1':M:Zeros(UInt(E)-1))

If required, the cycle count is rounded to a multiple of $2^{(E-1)}$ towards zero before being encoded.

A value of all ones in both the mantissa and exponent indicates the cycle count value exceeded the size of the cycle counter.

The value in this field is only valid when BRBINFINJ_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINFINJ_EL1.CCU == 1.
  - BRBINFINJ_EL1.VALID == 0b00.
- Otherwise, access to this field is **RW**.

Bits [31:18]

Reserved, RES0.

LASTFAILED, bit [17]

When FEAT_TME is implemented:

Indicates transaction failure or cancellation.

LASTFAILED	Meaning
0b0	Indicates that no transactions in a non-prohibited region have failed or been canceled between the previous Branch record and this Branch record.
0b1	Indicates that at least one transaction in a non-prohibited region has failed or been canceled between the previous Branch record and this Branch record.

The value in this field is only valid when BRBINFINJ_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINFINJ_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

T, bit [16]

When FEAT_TME is implemented:

Transactional state.

T	Meaning
0b0	The branch or exception was not executed in Transactional state.
0b1	The branch or exception was executed in Transactional state.

The value in this field is only valid when BRBINFINJ_EL1.VALID == 0b10 or BRBINFINJ_EL1.VALID == 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINFINJ_EL1.VALID == 0b00.
  - BRBINFINJ_EL1.VALID == 0b01.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Bits [15:14]

Reserved, RES0.

TYPE, bits [13:8]

Branch type.

TYPE	Meaning
0b000000	Unconditional direct branch, excluding Branch with link.
0b000001	Indirect branch, excluding Branch with link, Return from subroutine, and Exception return.
0b000010	Direct Branch with link.
0b000011	Indirect Branch with link.
0b000101	Return from subroutine.
0b000111	Exception return.
0b001000	Conditional direct branch.
0b100001	Debug halt.
0b100010	Call.
0b100011	Trap.
0b100100	SError.
0b100110	Instruction debug.
0b100111	Data debug.
0b101010	Alignment.
0b101011	Inst Fault.
0b101100	Data Fault.
0b101110	IRQ.
0b101111	FIQ.
0b110000	IMPLEMENTATION DEFINED exception to EL3.
0b111001	Debug State Exit.

All other values are reserved.

The value in this field is only valid when BRBINFINJ_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINFINJ_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

EL, bits [7:6]

The Exception level at the target address.

EL	Meaning	Applies when
0b00	EL0.	
0b01	EL1.	
0b10	EL2.	
0b11	EL3.	When FEAT_BRBEv1p1 is implemented

The value in this field is only valid when BRBINFINJ_EL1.VALID == 0b11 or BRBINFINJ_EL1.VALID == 0b01.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINFINJ_EL1.VALID == 0b00.
  - BRBINFINJ_EL1.VALID == 0b10.
- Otherwise, access to this field is **RW**.

MPRED, bit [5]

Branch mispredict.

MPRED	Meaning
0b0	Branch was correctly predicted or the result of the prediction was not captured.
0b1	Branch was incorrectly predicted.

The value in this field is only valid when BRBINFINJ_EL1.VALID == 0b11 or BRBINFINJ_EL1.VALID == 0b10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINFINJ_EL1.VALID == 0b00.
  - BRBINFINJ_EL1.VALID == 0b01.
  - BRBINFINJ_EL1.TYPE[5] == 1.
- Otherwise, access to this field is **RW**.

Bits [4:2]

Reserved, RES0.

VALID, bits [1:0]

The Branch record is valid.

VALID	Meaning
0b00	This Branch record is not valid. The values of following fields are not valid: • BRBTGTINJ_EL1.ADDRESS. • BRBSRCINJ_EL1.ADDRESS. • BRBINFINJ_EL1.MPRED. • BRBINFINJ_EL1.LASTFAILED. • BRBINFINJ_EL1.T. • BRBINFINJ_EL1.EL. • BRBINFINJ_EL1.TYPE. • BRBINFINJ_EL1.CC. • BRBINFINJ_EL1.CCU.
0b01	This Branch record is valid. The values of following fields are not valid: • BRBSRCINJ_EL1.ADDRESS. • BRBINFINJ_EL1.T. • BRBINFINJ_EL1.MPRED.
0b10	This Branch record is valid. The values of following fields are not valid: • BRBTGTINJ_EL1.ADDRESS. • BRBINFINJ_EL1.EL.
0b11	This Branch record is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing BRBINFINJ_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBINFINJ_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBINFINJ_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBINFINJ_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBINFINJ_EL1;
```

```
MSR BRBINFINJ_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBINFINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBINFINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBINFINJ_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBINF<n>_EL1, Branch Record Buffer Information Register <n>, n = 0 - 31

The BRBINF<n>_EL1 characteristics are:

Purpose

The information for Branch record n + (BRBFCR_EL1.BANK × 32).

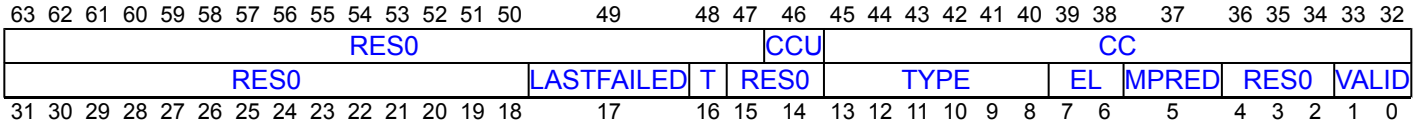
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBINF<n>_EL1 are UNDEFINED.

Attributes

BRBINF<n>_EL1 is a 64-bit register.

Field descriptions



Bits [63:47]

Reserved, RES0.

CCU, bit [46]

The number of PE clock cycles since the last Branch record entry is UNKNOWN.

CCU	Meaning
0b0	Indicates that the number of PE clock cycles since the last Branch record is indicated by BRBINF<n>_EL1.CC.
0b1	Indicates that the number of PE clock cycles since the last Branch record is UNKNOWN.

The value in this field is only valid when BRBINF<n>_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINF<n>_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RO**.

CC, bits [45:32]

The number of PE clock cycles since the last Branch record entry.

The format of this field uses a mantissa and exponent to express the cycle count value, as follows:

- CC bits[7:0] indicate the mantissa M.
- CC bits[13:8] indicate the exponent E.

The cycle count is expressed using the following function:

if IsZero(E) then UInt(M) else UInt('1':M:Zeros(UInt(E)-1))

If required, the cycle count is rounded to a multiple of $2^{(E-1)}$ towards zero before being encoded.

A value of all ones in both the mantissa and exponent indicates the cycle count value exceeded the size of the cycle counter.

The value in this field is only valid when BRBINF<n>_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINF<n>_EL1.CCU == 1.
  - BRBINF<n>_EL1.VALID == 0b00.
- Otherwise, access to this field is **RO**.

Bits [31:18]

Reserved, RES0.

LASTFAILED, bit [17]

When FEAT_TME is implemented:

Indicates transaction failure or cancellation.

LASTFAILED	Meaning
0b0	Indicates that no transactions in a non-prohibited region have failed or been canceled between the previous Branch record and this Branch record.
0b1	Indicates that at least one transaction in a non-prohibited region has failed or been canceled between the previous Branch record and this Branch record.

The value in this field is only valid when BRBINF<n>_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINF<n>_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

T, bit [16]

When FEAT_TME is implemented:

Transactional state.

T	Meaning
0b0	The branch or exception was not executed in Transactional state.
0b1	The branch or exception was executed in Transactional state.

The value in this field is only valid when BRBINF<n>_EL1.VALID == 0b10 or BRBINF<n>_EL1.VALID == 0b11.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINF<n>_EL1.VALID == 0b00.
  - BRBINF<n>_EL1.VALID == 0b01.
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [15:14]

Reserved, RES0.

TYPE, bits [13:8]

Branch type.

TYPE	Meaning
0b000000	Unconditional direct branch, excluding Branch with link.
0b000001	Indirect branch, excluding Branch with link, Return from subroutine, and Exception return.
0b000010	Direct Branch with link.
0b000011	Indirect Branch with link.
0b000101	Return from subroutine.
0b000111	Exception return.
0b001000	Conditional direct branch.
0b100001	Debug halt.
0b100010	Call.
0b100011	Trap.
0b100100	SError.
0b100110	Instruction debug.
0b100111	Data debug.
0b101010	Alignment.
0b101011	Inst Fault.
0b101100	Data Fault.
0b101110	IRQ.
0b101111	FIQ.
0b110000	IMPLEMENTATION DEFINED exception to EL3.
0b111001	Debug State Exit.

All other values are reserved.

The value in this field is only valid when BRBINF<n>_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINF<n>_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RO**.

EL, bits [7:6]

The Exception level at the target address.

EL	Meaning	Applies when
0b00	EL0.	
0b01	EL1.	
0b10	EL2.	
0b11	EL3.	When FEAT_BRBEv1p1 is implemented

The value in this field is only valid when BRBINF<n>_EL1.VALID == 0b11 or BRBINF<n>_EL1.VALID == 0b01.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINF<n>_EL1.VALID == 0b00.
  - BRBINF<n>_EL1.VALID == 0b10.
- Otherwise, access to this field is **RO**.

MPRED, bit [5]

Branch mispredict.

MPRED	Meaning
0b0	Branch was correctly predicted or the result of the prediction was not captured.
0b1	Branch was incorrectly predicted.

The value in this field is only valid when BRBINF<n>_EL1.VALID == 0b11 or BRBINF<n>_EL1.VALID == 0b10.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINF<n>_EL1.VALID == 0b00.
  - BRBINF<n>_EL1.VALID == 0b01.
  - BRBINF<n>_EL1.TYPE[5] == 1.
- Otherwise, access to this field is **RO**.

Bits [4:2]

Reserved, RES0.

VALID, bits [1:0]

The Branch record is valid.

VALID	Meaning
0b00	This Branch record is not valid. The values of following fields are not valid: • BRBTGT<n>_EL1.ADDRESS. • BRBSRC<n>_EL1.ADDRESS. • BRBINF<n>_EL1.MPRED. • BRBINF<n>_EL1.LASTFAILED. • BRBINF<n>_EL1.T. • BRBINF<n>_EL1.EL. • BRBINF<n>_EL1.TYPE. • BRBINF<n>_EL1.CC. • BRBINF<n>_EL1.CCU.
0b01	This Branch record is valid. The values of following fields are not valid: • BRBSRC<n>_EL1.ADDRESS. • BRBINF<n>_EL1.T. • BRBINF<n>_EL1.MPRED.
0b10	This Branch record is valid. The values of following fields are not valid: • BRBTGT<n>_EL1.ADDRESS. • BRBINF<n>_EL1.EL.
0b11	This Branch record is valid.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing BRBINF<n>_EL1

BRBINF<n>_EL1 reads-as-zero if n + (BRBFCR_EL1.BANK × 32) >= BRBIDR0_EL1.NUMREC.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBINF<m>_EL1 ; Where m = 0-31
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1000	m[3:0]	m[4]:0b00

```
integer m = UInt(op2<2>:CRm<3:0>);

if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBINF_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBINF_EL1[m];
elsif PSTATE.EL == EL3 then
    if m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBINF_EL1[m];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBSRCINJ_EL1, Branch Record Buffer Source Address Injection Register

The BRBSRCINJ_EL1 characteristics are:

Purpose

The source address of a Branch record for injection.

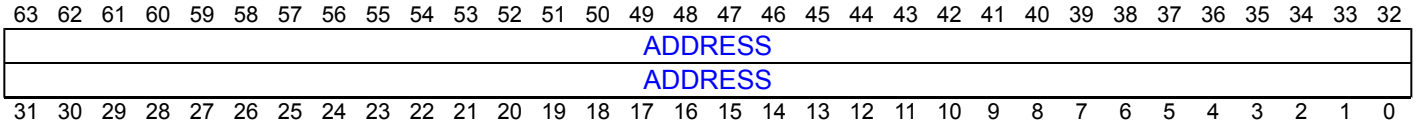
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBSRCINJ_EL1 are UNDEFINED.

Attributes

BRBSRCINJ_EL1 is a 64-bit register.

Field descriptions



ADDRESS, bits [63:0]

Source virtual address of the Branch record.

When a direct write occurs with a value with ADDRESS bits [63:P] indicating an invalid address, an UNKNOWN value which indicates an invalid address is written to bits [63:P].

An invalid address is:

- For an address in a translation regime with 2 VA ranges, with bits [63:P] not all zeroes or all ones.
- For an address in a translation regime with a single VA range, with bits [63:P] not all zeroes.

P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The value in bits [P-1:0] is the value written.

When a direct write occurs with a value with ADDRESS bits [63:P] indicating a valid address, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINFINJ_EL1.VALID == 0b00.
  - BRBINFINJ_EL1.VALID == 0b01.
- Otherwise, access to this field is **RW**.

Accessing BRBSRCINJ_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBSRCINJ_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBSRCINJ_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBSRCINJ_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBSRCINJ_EL1;
```

```
MSR BRBSRCINJ_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBSRCINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBSRCINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBSRCINJ_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBSRC<n>_EL1, Branch Record Buffer Source Address Register <n>, n = 0 - 31

The BRBSRC<n>_EL1 characteristics are:

Purpose

The source address of Branch record n + (BRBFCR_EL1.BANK × 32).

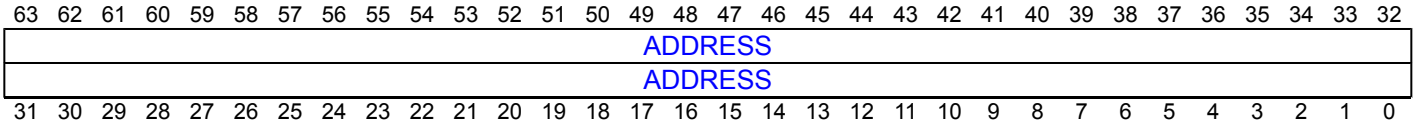
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBSRC<n>_EL1 are UNDEFINED.

Attributes

BRBSRC<n>_EL1 is a 64-bit register.

Field descriptions



ADDRESS, bits [63:0]

Source virtual address of the Branch record.

When an indirect write occurs with a value with ADDRESS bits [63:P] indicating an invalid address, an UNKNOWN value which indicates an invalid address is written to bits [63:P].

An invalid address is:

- For an address in a translation regime with 2 VA ranges, with bits [63:P] not all zeroes or all ones.
- For an address in a translation regime with a single VA range, with bits [63:P] not all zeroes.

P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The value in bits [P-1:0] is the value written.

When an indirect write occurs with a value with ADDRESS bits [63:P] indicating a valid address, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINF<n>_EL1.VALID == 0b00.
  - BRBINF<n>_EL1.VALID == 0b01.
- Otherwise, access to this field is **RO**.

Accessing BRBSRC<n>_EL1

BRBSRC<n>_EL1 is RES0 if n + (BRBFCR_EL1.BANK × 32) >= BRBIDR0_EL1.NUMREC.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBSRC<m>_EL1 ; Where m = 0-31
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1000	m[3:0]	m[4]:0b01

```
integer m = UInt(op2<2>:CRm<3:0>);

if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBSRC_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBSRC_EL1[m];
elsif PSTATE.EL == EL3 then
    if m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBSRC_EL1[m];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBTGTINJ_EL1, Branch Record Buffer Target Address Injection Register

The BRBTGTINJ_EL1 characteristics are:

Purpose

The target address of a Branch record for injection.

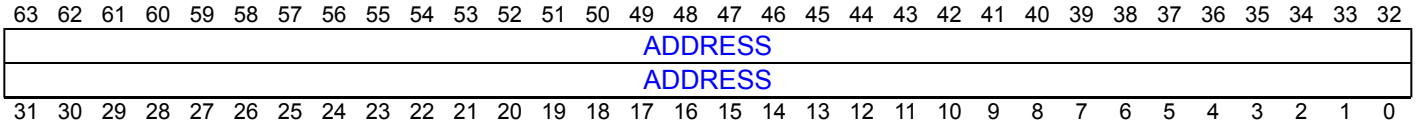
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBTGTINJ_EL1 are UNDEFINED.

Attributes

BRBTGTINJ_EL1 is a 64-bit register.

Field descriptions



ADDRESS, bits [63:0]

Target virtual address of the Branch record.

When a direct write occurs with a value with ADDRESS bits [63:P] indicating an invalid address, an UNKNOWN value which indicates an invalid address is written to bits [63:P].

An invalid address is:

- For an address in a translation regime with 2 VA ranges, with bits [63:P] not all zeroes or all ones.
- For an address in a translation regime with a single VA range, with bits [63:P] not all zeroes.

P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The value in bits [P-1:0] is the value written.

When a direct write occurs with a value with ADDRESS bits [63:P] indicating a valid address, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINFINJ_EL1.VALID == 0b00.
  - BRBINFINJ_EL1.VALID == 0b10.
- Otherwise, access to this field is **RW**.

Accessing BRBTGTINJ_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBTGTINJ_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBTGTINJ_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBTGTINJ_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBTGTINJ_EL1;
```

```
MSR BRBTGTINJ_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTGTINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTGTINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBTGTINJ_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBTGT<n>_EL1, Branch Record Buffer Target Address Register <n>, n = 0 - 31

The BRBTGT<n>_EL1 characteristics are:

Purpose

The target address of Branch record n + (BRBFCR_EL1.BANK × 32).

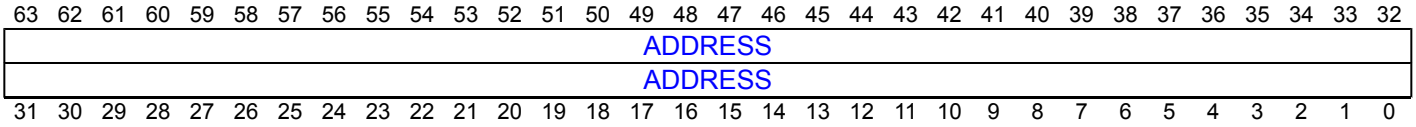
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBTGT<n>_EL1 are UNDEFINED.

Attributes

BRBTGT<n>_EL1 is a 64-bit register.

Field descriptions



ADDRESS, bits [63:0]

Target virtual address of the Branch record.

When an indirect write occurs with a value with ADDRESS bits [63:P] indicating an invalid address, an UNKNOWN value which indicates an invalid address is written to bits [63:P].

An invalid address is:

- For an address in a translation regime with 2 VA ranges, with bits [63:P] not all zeroes or all ones.
- For an address in a translation regime with a single VA range, with bits [63:P] not all zeroes.

P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The value in bits [P-1:0] is the value written.

When an indirect write occurs with a value with ADDRESS bits [63:P] indicating a valid address, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if any the following are true:
  - BRBINF<n>_EL1.VALID == 0b00.
  - BRBINF<n>_EL1.VALID == 0b10.
- Otherwise, access to this field is **RO**.

Accessing BRBTGT<n>_EL1

BRBTGT<n>_EL1 is RES0 if n + (BRBFCR_EL1.BANK × 32) >= BRBIDR0_EL1.NUMREC.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBTGT<m>_EL1 ; Where m = 0-31
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1000	m[3:0]	m[4]:0b10

```
integer m = UInt(op2<2>:CRm<3:0>);

if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBTGT_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBTGT_EL1[m];
elsif PSTATE.EL == EL3 then
    if m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBTGT_EL1[m];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBTS_EL1, Branch Record Buffer Timestamp Register

The BRBTS_EL1 characteristics are:

Purpose

Captures the Timestamp value on a BRBE freeze event.

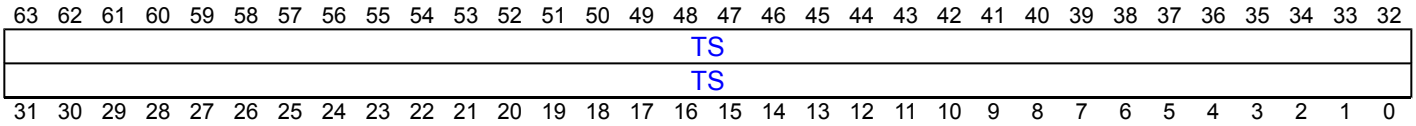
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBTS_EL1 are UNDEFINED.

Attributes

BRBTS_EL1 is a 64-bit register.

Field descriptions



TS, bits [63:0]

Timestamp value at the time of a BRBE freeze event.

The reset behavior of this field is:

- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to '0000000000000000000000000000000000000000000000000000000000000000'.

When FEAT_BRBEv1p1 is implemented, Arm recommends that this field is preserved on a Warm reset, but it is IMPLEMENTATION DEFINED whether this field resets to 0 or is preserved.

When FEAT_BRBEv1p1 is implemented, on a Cold reset it is IMPLEMENTATION DEFINED whether this field resets to an architecturally UNKNOWN value.

Accessing BRBTS_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBTS_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBTS_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBTS_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBTS_EL1;
```

```
MSR BRBTS_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTS_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTS_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBTS_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CCSIDR2_EL1, Current Cache Size ID Register 2

The CCSIDR2_EL1 characteristics are:

Purpose

Provides the information about the architecture of the currently selected cache from bits[63:32] of CCSIDR_EL1.

Configuration

AArch64 System register CCSIDR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register CCSIDR2[31:0].

This register is present only when FEAT_CCIDX is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CCSIDR2_EL1 are UNDEFINED.

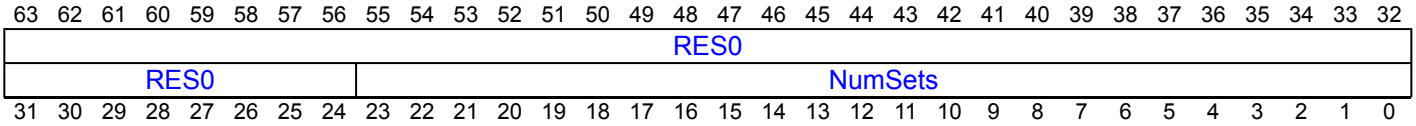
In an implementation which does not support AArch32 at EL1, it is IMPLEMENTATION DEFINED whether reading this register gives an UNKNOWN value or is UNDEFINED.

The implementation includes one CCSIDR2_EL1 for each cache that it can access. CSSELR_EL1 selects which Cache Size ID Register is accessible.

Attributes

CCSIDR2_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

NumSets, bits [23:0]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Accessing CCSIDR2_EL1

If CSSELR_EL1.{TnD, Level, InD} is programmed to a cache level that is not implemented, then on a read of the CCSIDR2_EL1 the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR2_EL1 read is treated as NOP.
- The CCSIDR2_EL1 read is UNDEFINED. If FEAT_IDST is implemented, this is permitted to be reported with EC syndrome value `0x18`.
- The CCSIDR2_EL1 read returns an UNKNOWN value.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CCSIDR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b001	0b0000	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_CCIDX) && IsFeatureImplemented(FEAT_AA64)) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_EVT) && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = CCSIDR2_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CCSIDR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CCSIDR2_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CCSIDR_EL1, Current Cache Size ID Register

The CCSIDR_EL1 characteristics are:

Purpose

Provides information about the architecture of the currently selected cache.

Configuration

AArch64 System register CCSIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CCSIDR[31:0].

AArch64 System register CCSIDR_EL1 bits [63:32] are architecturally mapped to AArch32 System register CCSIDR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CCSIDR_EL1 are UNDEFINED.

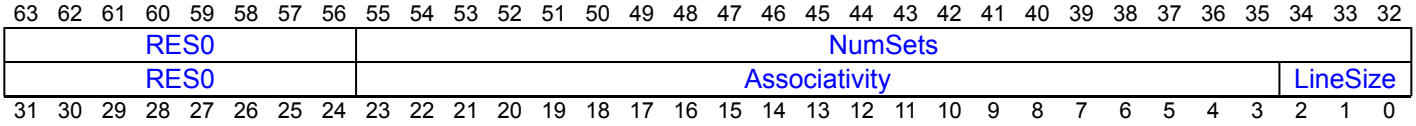
The implementation includes one CCSIDR_EL1 for each cache that it can access. CSSELR_EL1 selects which Cache Size ID Register is accessible.

Attributes

CCSIDR_EL1 is a 64-bit register.

Field descriptions

When FEAT_CCIDX is implemented:



Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [63:56]

Reserved, RES0.

NumSets, bits [55:32]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Bits [31:24]

Reserved, RES0.

Associativity, bits [23:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log_2(Number of bytes in cache line)) - 4. For example:

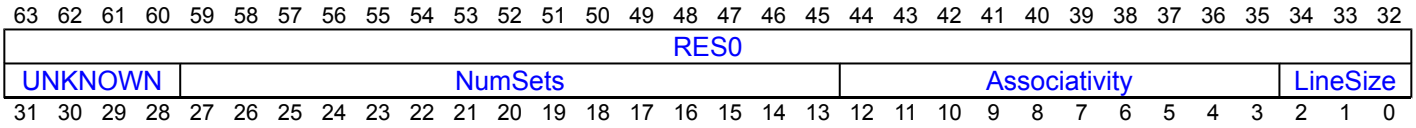
- For a line length of 16 bytes: $Log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.
- For a line length of 32 bytes: $Log_2(32) = 5$, LineSize entry = 1.

Note

The C++ 17 specification has two defined parameters relating to the granularity of memory that does not interfere. For generic software and tools, Arm will set the hardware_destructive_interference_size parameter to 256 bytes and the hardware_constructive_interference_size parameter to 64 bytes.

When FEAT_MTE2 is implemented, where a cache only holds Allocation tags, this field is RES0.

Otherwise:



Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [63:32]

Reserved, RES0.

Bits [31:28]

Reserved, UNKNOWN.

NumSets, bits [27:13]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Associativity, bits [12:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log_2(Number of bytes in cache line)) - 4. For example:

- For a line length of 16 bytes: $Log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.
- For a line length of 32 bytes: $Log_2(32) = 5$, LineSize entry = 1.

When FEAT_MTE2 is implemented, where a cache only holds Allocation tags, this field is RES0.

Note

The C++ 17 specification has two defined parameters relating to the granularity of memory that does not interfere. For generic software and tools, Arm will set the

hardware_destructive_interference_size parameter to 256 bytes and the hardware_constructive_interference_size parameter to 64 bytes.

Accessing CCSIDR_EL1

If CSSELR_EL1.{TnD, Level, InD} is programmed to a cache level that is not implemented, then on a read of the CCSIDR_EL1 the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR_EL1 read is treated as NOP.
- The CCSIDR_EL1 read is UNDEFINED. If FEAT_IDST is implemented, this is permitted to be reported with EC syndrome value `0x18`.
- The CCSIDR_EL1 read returns an UNKNOWN value.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CCSIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b001	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_EVT) && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.CCSIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = CCSIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CCSIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CCSIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CFP RCTX, Control Flow Prediction Restriction by Context

The CFP RCTX characteristics are:

Purpose

Control Flow Prediction Restriction by Context applies to all Control Flow Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

Control flow predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control speculative execution occurring after the instruction is complete and synchronized.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

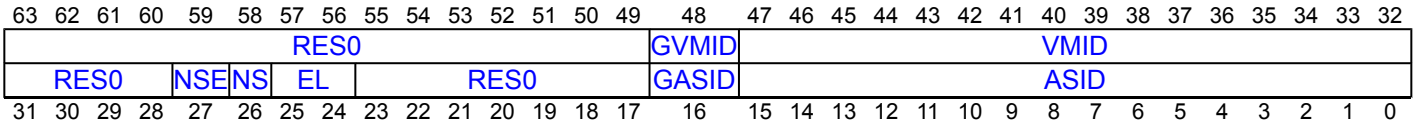
Configuration

This instruction is present only when FEAT_SPECRES is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CFP RCTX are UNDEFINED.

Attributes

CFP RCTX is a 64-bit System instruction.

Field descriptions



Bits [63:49]

Reserved, RES0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 target execution context.
0b1	Applies to all VMIDs for an EL0 or EL1 target execution context.

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and the target execution context is either:

- EL1.
- EL0 when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, this field is treated as the current VMID.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

If the implementation supports 16 bits of VMID, then the upper 8 bits of the VMID must be written to 0 by software when the context being affected only uses 8 bits.

Bits [31:28]

Reserved, RES0.

NSE, bit [27]

When FEAT_RME is implemented:

Together with the NS field, selects the Security state.

For a description of the values derived by evaluating NS and NSE together, see CFP_RCTX.NS.

Otherwise:

Reserved, RES0.

NS, bit [26]

When FEAT_RME is implemented:

Together with the NSE field, selects the Security state.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

Some Effective values are determined by the current Security state:

- When executed in Secure state, the Effective value of NSE is 0.
- When executed in Non-secure state, the Effective value of {NSE, NS} is {0, 1}.
- When executed in Realm state, the Effective value of {NSE, NS} is {1, 1}.

This instruction is treated as a NOP when executed at EL3 and either:

- CFP_RCTX.{NSE, NS} selects a reserved value.
- CFP_RCTX.{NSE, NS} == {1, 0} and CFP_RCTX.EL has a value other than 0b11.

Otherwise:

Security State.

NS	Meaning
0b0	Secure state.
0b1	Non-secure state.

When executed in Non-secure state, the Effective value of NS is 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

If the instruction is executed at an Exception level lower than the specified level, or is specified to apply to a combination of Exception level and Security state that is not implemented, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

GASID	Meaning
0b0	Applies to specified ASID for an EL0 target execution context.
0b1	Applies to all ASIDs for an EL0 target execution context.

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [15:0]

Only applies for an EL0 target execution context and when bit[16] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being affected only uses 8 bits.

Executing CFP RCTX

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
CFP RCTX, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0011	0b100

```
if !(IsFeatureImplemented(FEAT_SPECRES) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.CFPRCTX == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnRCTX == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.RestrictPrediction(X[t, 64], RestrictType_ControlFlow);
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.CFPRCTX == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.RestrictPrediction(X[t, 64], RestrictType_ControlFlow);
elsif PSTATE.EL == EL2 then
    AArch64.RestrictPrediction(X[t, 64], RestrictType_ControlFlow);
elsif PSTATE.EL == EL3 then
    AArch64.RestrictPrediction(X[t, 64], RestrictType_ControlFlow);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CLIDR_EL1, Cache Level ID Register

The CLIDR_EL1 characteristics are:

Purpose

Identifies the type of cache, or caches, that are implemented at each level and can be managed using the architected cache maintenance instructions that operate by set/way, up to a maximum of seven levels. Also identifies the Level of Coherence (LoC) and Level of Unification (LoU) for the cache hierarchy.

Configuration

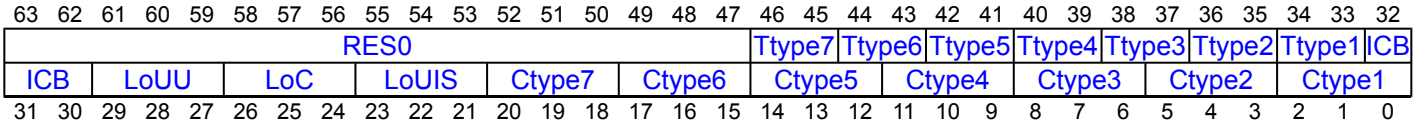
AArch64 System register CLIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CLIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CLIDR_EL1 are UNDEFINED.

Attributes

CLIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:47]

Reserved, RES0.

Ttype<n>, bits [2(n-1)+34:2(n-1)+33], for n = 7 to 1

When FEAT_MTE2 is implemented:

Tag cache type. Indicate the type of cache that is implemented and can be managed using the architected cache maintenance instructions that operate by set/way at each level, from Level 1 up to a maximum of seven levels of cache hierarchy.

Ttype<n>	Meaning
0b00	No Tag Cache.
0b01	Separate Allocation Tag Cache.
0b10	Unified Allocation Tag and Data cache, Allocation Tags and Data in unified lines.
0b11	Unified Allocation Tag and Data cache, Allocation Tags and Data in separate lines.

Otherwise:

Reserved, RES0.

ICB, bits [32:30]

Inner cache boundary. This field indicates the boundary for caching Inner Cacheable memory regions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ICB	Meaning
0b000	Not disclosed by this mechanism.
0b001	L1 cache is the highest Inner Cacheable level.
0b010	L2 cache is the highest Inner Cacheable level.
0b011	L3 cache is the highest Inner Cacheable level.
0b100	L4 cache is the highest Inner Cacheable level.
0b101	L5 cache is the highest Inner Cacheable level.
0b110	L6 cache is the highest Inner Cacheable level.
0b111	L7 cache is the highest Inner Cacheable level.

Access to this field is **RO**.

LoUU, bits [29:27]

Level of Unification Uniprocessor for the cache hierarchy.

For a description of the values of this field, see Terminology for Clean, Invalidate, and Clean and Invalidate instructions.

Note

This field does not describe the requirements for instruction cache invalidation. See CTR_EL0.DIC.

Note

When FEAT_S2FWB is implemented, the architecture requires that this field is zero so that no levels of data cache need to be cleaned in order to manage coherency with instruction fetches.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

LoC, bits [26:24]

Level of Coherence for the cache hierarchy.

For a description of the values of this field, see Terminology for Clean, Invalidate, and Clean and Invalidate instructions.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

LoUIS, bits [23:21]

Level of Unification Inner Shareable for the cache hierarchy.

For a description of the values of this field, see Terminology for Clean, Invalidate, and Clean and Invalidate instructions.

Note

This field does not describe the requirements for instruction cache invalidation. See CTR_EL0.DIC.

Note

When FEAT_S2FWB is implemented, the architecture requires that this field is zero so that no levels of data cache need to be cleaned in order to manage coherency with instruction fetches.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Ctype<n>, bits [3(n-1)+2:3(n-1)], for n = 7 to 1

Cache Type fields. Indicate the type of cache that is implemented and can be managed using the architected cache maintenance instructions that operate by set/way at each level, from Level 1 up to a maximum of seven levels of cache hierarchy. Possible values of each field are:

Ctype<n>	Meaning
0b000	No cache.
0b001	Instruction cache only.
0b010	Data cache only.
0b011	Separate instruction and data caches.
0b100	Unified cache.

All other values are reserved.

If software reads the Cache Type fields from Ctype1 upwards, once it has seen a value of 0b000, no caches that can be managed using the architected cache maintenance instructions that operate by set/way exist at further-out levels of the hierarchy. So, for example, if Ctype3 is the first Cache Type field with a value of 0b000, the values of Ctype4 to Ctype7 must be ignored.

Accessing CLIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CLIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b001	0b0000	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_EVT) && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.CLIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = CLIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CLIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CLIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTFRQ_EL0, Counter-timer Frequency Register

The CNTFRQ_EL0 characteristics are:

Purpose

This register is provided so that software can discover the frequency of the system counter. It must be programmed with this value as part of system initialization. The value of the register is not interpreted by hardware.

Configuration

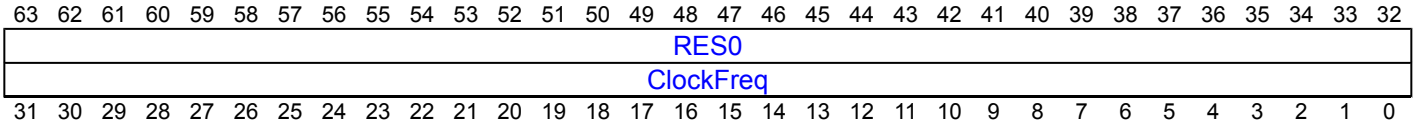
AArch64 System register CNTFRQ_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTFRQ[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTFRQ_EL0 are UNDEFINED.

Attributes

CNTFRQ_EL0 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

ClockFreq, bits [31:0]

Clock frequency. Indicates the system counter clock frequency, in Hz.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFRQ_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTFRQ_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.<EL0PCTEN,EL0VCTEN> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.<EL0PCTEN,EL0VCTEN> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = CNTFRQ_EL0;
elsif PSTATE.EL == EL1 then
    X[t, 64] = CNTFRQ_EL0;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTFRQ_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTFRQ_EL0;
```

```
MSR CNTFRQ_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    CNTFRQ_EL0 = X[t, 64];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHCTL_EL2, Counter-timer Hypervisor Control Register

The CNTHCTL_EL2 characteristics are:

Purpose

Controls the generation of an event stream from the physical counter, and access from EL1 to the physical counter and the EL1 physical timer.

Configuration

AArch64 System register CNTHCTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHCTL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHCTL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

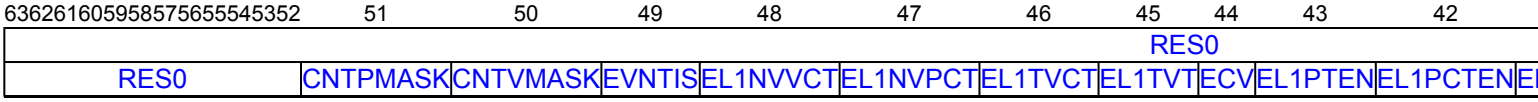
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

CNTHCTL_EL2 is a 64-bit register.

Field descriptions

When ELIsInHost(EL2):



Bits [63:20]

Reserved, RES0.

CNTPMASK, bit [19]

When FEAT_RME is implemented:

CNTPMASK	Meaning
0b0	This control has no effect on CNTP_CTL_EL0.IMASK.
0b1	CNTP_CTL_EL0.IMASK behaves as if set to 1 for all purposes other than a direct read of the field.

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CNTVMASK, bit [18]

When FEAT_RME is implemented:

CNTVMASK	Meaning
0b0	This control has no effect on CNTV_CTL_EL0.IMASK.
0b1	CNTV_CTL_EL0.IMASK behaves as if set to 1 for all purposes other than a direct read of the field.

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVNTIS, bit [17]

When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

EVNTIS	Meaning
0b0	The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[15:0].
0b1	The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[23:8].

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVVCT, bit [16]

When FEAT_ECV is implemented:

When HCR_EL2.TGE is 0 and the Effective value of HCR_EL2.{NV2, NV1, NV} is {1, 0, 1}, traps EL1 accesses to the specified EL1 virtual timer registers using the EL02 descriptors to EL2 as follows:

Accesses to CNTV_CTL_EL02 and CNTV_CVAL_EL02 are trapped to EL2 and reported with EC syndrome value 0x18.

EL1NVVCT	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the specified registers are trapped to EL2.

If HCR_EL2.TGE is 1 or the Effective value of HCR_EL2.{NV2, NV1, NV} is not {1, 0, 1}, this control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVPCT, bit [15]

When FEAT_ECV is implemented:

When HCR_EL2.TGE is 0 and the Effective value of HCR_EL2.{NV2, NV1, NV} is {1, 0, 1}, traps EL1 accesses to the specified EL1 physical timer registers using the EL02 descriptors to EL2 as follows:

Accesses to CNTP_CTL_EL02 and CNTP_CVAL_EL02 are trapped to EL2 and reported with EC syndrome value `0x18`.

EL1NVPCT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL1 accesses to the specified registers are trapped to EL2.

If HCR_EL2.TGE is 1 or the Effective value of HCR_EL2.{NV2, NV1, NV} is not {1, 0, 1}, this control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVCT, bit [14]

When FEAT_ECV is implemented:

Traps EL0 and EL1 accesses to the EL1 virtual counter registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to CNTVCT_EL0 and CNTVCTSS_EL0 are trapped to EL2 and reported using EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0VCTEN.
- In AArch32 state, accesses to CNTVCT are trapped to EL2 and reported with EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0VCTEN or CNTKCTL.PL0VCTEN.

EL1TVCT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 and EL1 accesses to the specified registers are trapped to EL2.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

If HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVT, bit [13]

When FEAT_ECV is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps EL0 and EL1 accesses to the EL1 virtual timer registers to EL2, when EL2 is enabled for the current Security state, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0VTEN:
  - CNTV_CTL_EL0.
  - CNTV_CVAL_EL0.
  - CNTV_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped to EL2 and reported using EC syndrome value `0x03`, and MCRR and MRRC accesses are trapped to EL2 and reported using EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0VTEN or CNTKCTL.PL0VTEN:
  - CNTV_CTL.
  - CNTV_CVAL.
  - CNTV_TVAL.

EL1TVT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 and EL1 accesses to the specified registers are trapped to EL2.

If HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECV, bit [12]

When FEAT_ECV_POFF is implemented:

Enables the Enhanced Counter Virtualization functionality registers.

When the Enhanced Counter Virtualization is enabled, the behavior is as follows:

- An MRS to CNTPCT_EL0 from either EL0 or EL1 that is not trapped will return the value (PCount<63:0> - CNTPOFF_EL2<63:0>).
- The EL1 physical timer interrupt is triggered when ((PCount<63:0> - CNTPOFF_EL2<63:0>) - PCVal<63:0>) is greater than or equal to 0.

PCount<63:0> is the physical count returned when CNTPCT_EL0 is read from EL2 or EL3.

PCVal<63:0> is the EL1 physical timer compare value for this timer.

ECV	Meaning
`0b0`	Enhanced Counter Virtualization functionality is disabled.
`0b1`	Enhanced Counter Virtualization functionality is enabled.

When HCR_EL2.TGE is 1 or SCR_EL3.{NS, EEL2} is {0, 0}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1PTEN, bit [11]

When HCR_EL2.TGE is 0, traps EL0 and EL1 accesses to the EL1 physical timer registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0PTEN:
  - CNTP_CTL_EL0.
  - CNTP_CVAL_EL0.
  - CNTP_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03` and MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0PTEN or CNTKCTL.PL0PTEN:
  - CNTP_CTL.
  - CNTP_CVAL.
  - CNTP_TVAL.

EL1PTEN	Meaning
`0b0`	EL0 and EL1 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL1PCTEN, bit [10]

When HCR_EL2.TGE is 0, traps EL0 and EL1 accesses to the EL1 physical counter registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to CNTPCT_EL0 and CNTPCTSS_EL0 are trapped to EL2 and reported with EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0PCTEN.
- In AArch32 state, MRRC or MCRR accesses to CNTPCT are trapped to EL2 and reported with EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0PCTEN or CNTKCTL.PL0PCTEN.

EL1PCTEN	Meaning
`0b0`	EL0 and EL1 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0PTEN, bit [9]

When HCR_EL2.TGE is 1, traps EL0 accesses to the physical timer registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`:
  - CNTP_CTL_EL0.
  - CNTP_CVAL_EL0.
  - CNTP_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03` and MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`:
  - CNTP_CTL.
  - CNTP_CVAL.
  - CNTP_TVAL.

EL0PTEN	Meaning
`0b0`	EL0 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 0, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0VTEN, bit [8]

When HCR_EL2.TGE is 1, traps EL0 accesses to the virtual timer registers to EL2 as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`:
  - CNTV_CTL_EL0.
  - CNTV_CVAL_EL0.
  - CNTV_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03` and MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`:
  - CNTV_CTL.
  - CNTV_CVAL.
  - CNTV_TVAL.

EL0VTEN	Meaning
`0b0`	EL0 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 0, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTI, bits [7:4]

Selects which bit of CNTPCT_EL0, as seen from EL2, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTHCTL_EL2.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of CNTPCT_EL0.

Otherwise, this field selects a trigger bit in the range 0 to 15 of CNTPCT_EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the CNTPCT_EL0 trigger bit, as seen from EL2 and defined by EVNTI, generates an event when the event stream is enabled.

EVNTDIR	Meaning
`0b0`	A 0 to 1 transition of the trigger bit triggers an event.
`0b1`	A 1 to 0 transition of the trigger bit triggers an event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

Enables the generation of an event stream from CNTPCT_EL0 as seen from EL2.

EVNTEN	Meaning
`0b0`	Disables the event stream.
`0b1`	Enables the event stream.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0VCTEN, bit [1]

When HCR_EL2.TGE is 1, traps EL0 accesses to the frequency register and virtual counter registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`:
  - CNTVCT_EL0.
  - CNTVCTSS_EL0.
  - CNTFRQ_EL0 if CNTHCTL_EL2.EL0PCTEN is 0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03` and MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`:
  - CNTVCT and if CNTHCTL.EL0PCTEN is 0, CNTFRQ.

EL0VCTEN	Meaning
`0b0`	EL0 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE is 0, the field is ignored for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0PCTEN, bit [0]

Traps EL0 accesses to the frequency register and physical counter registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`:
  - CNTPCT_EL0.
  - CNTPCTSS_EL0.
  - CNTFRQ_EL0 if CNTHCTL_EL2.EL0VCTEN is 0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03` and MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`:
  - CNTPCT and if CNTHCTL_EL2.EL0VCTEN is 0, CNTFRQ.

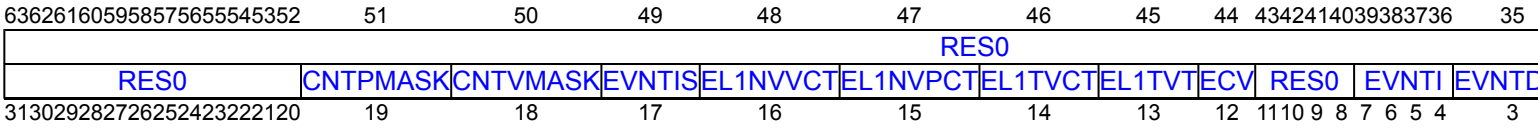
EL0PCTEN	Meaning
`0b0`	From AArch64 state: EL0 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE is 0, the control does not cause any instructions to be trapped for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:



The following field descriptions apply in all Armv8.0 implementations.

The descriptions also explain the behavior when EL3 is implemented and EL2 is not implemented.

Bits [63:20]

Reserved, RES0.

CNTPMASK, bit [19]

When FEAT_RME is implemented:

CNTPMASK	Meaning
`0b0`	This control has no effect on CNTP_CTL_EL0.IMASK.
`0b1`	CNTP_CTL_EL0.IMASK behaves as if set to 1 for all purposes other than a direct read of the field.

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CNTVMASK, bit [18]
When FEAT_RME is implemented:

CNTVMASK	Meaning
0b0	This control has no effect on CNTV_CTL_EL0.IMASK.
0b1	CNTV_CTL_EL0.IMASK behaves as if set to 1 for all purposes other than a direct read of the field.

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVNTIS, bit [17]
When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

EVNTIS	Meaning
0b0	The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[15:0].
0b1	The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[23:8].

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVVCT, bit [16]
When FEAT_ECV is implemented:

When the Effective value of HCR_EL2.{NV2, NV1, NV} is {1, 0, 1}, traps EL1 accesses to the specified EL1 virtual timer registers using the EL02 descriptors to EL2 as follows:

Accesses to CNTV_CTL_EL02 and CNTV_CVAL_EL02 are trapped to EL2 and reported with EC syndrome value 0x18.

EL1NVVCT	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the specified registers are trapped to EL2.

If the Effective value of HCR_EL2.{NV2, NV1, NV} is not {1, 0, 1}, this control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVPCT, bit [15]
When FEAT_ECV is implemented:

When the Effective value of HCR_EL2.{NV2, NV1, NV} is {1, 0, 1}, traps EL1 accesses to the specified EL1 physical timer registers using the EL02 descriptors to EL2 as follows:

Accesses to CNTP_CTL_EL02 and CNTP_CVAL_EL02 are trapped to EL2 and reported with EC syndrome value `0x18`.

EL1NVPCT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL1 accesses to the specified registers are trapped to EL2.

If the Effective value of HCR_EL2.{NV2, NV1, NV} is not {1, 0, 1}, this control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVCT, bit [14]
When FEAT_ECV is implemented:

Traps EL0 and EL1 accesses to the EL1 virtual counter registers to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to CNTVCT_EL0 and CNTVCTSS_EL0 are trapped to EL2 and reported using EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0VCTEN.
- In AArch32 state, accesses to CNTVCT are trapped to EL2 and reported using EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0VCTEN or CNTKCTL.PL0VCTEN.

EL1TVCT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 and EL1 accesses to the specified registers are trapped to EL2.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVT, bit [13]
When FEAT_ECV is implemented:

If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then traps EL0 and EL1 accesses to the EL1 virtual timer registers to EL2, when EL2 is enabled for the current Security state, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0VTEN:
  - CNTV_CTL_EL0.
  - CNTV_CVAL_EL0.
  - CNTV_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03` and MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0VTEN or CNTKCTL.PL0VTEN:
  - CNTV_CTL.
  - CNTV_CVAL.
  - CNTV_TVAL.

EL1TVT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 and EL1 accesses to the specified registers are trapped to EL2.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECV, bit [12]
When FEAT_ECV_POFF is implemented:

Enables the Enhanced Counter Virtualization functionality registers.

When the Enhanced Counter Virtualization is enabled, the behavior is as follows:

- An MRS to CNTPCT_EL0 from either EL0 or EL1 that is not trapped will return the value (PCount<63:0> - CNTPOFF_EL2<63:0>).
- The EL1 physical timer interrupt is triggered when ((PCount<63:0> - CNTPOFF_EL2<63:0>) - PCVal<63:0>) is greater than or equal to 0.

PCount is the physical count returned when CNTPCT_EL0 is read from EL2 or EL3.

PCVal<63:0> is the EL1 physical timer compare value for this timer.

ECV	Meaning
`0b0`	Enhanced Counter Virtualization functionality is disabled.
`0b1`	Enhanced Counter Virtualization functionality is enabled.

When SCR_EL3.{NS, EEL2} is {0, 0} or if FEAT_ECV_POFF is not implemented, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [11:8]

Reserved, RES0.

EVNTI, bits [7:4]

Selects which bit of CNTPCT_EL0, as seen from EL2,is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTHCTL_EL2.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of CNTPCT_EL0.

Otherwise, this field selects a trigger bit in the range 0 to 15 of CNTPCT_EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the CNTPCT_EL0 trigger bit, as seen from EL2 and defined by EVNTI, generates an event when the event stream is enabled.

EVNTDIR	Meaning
0b0	A 0 to 1 transition of the trigger bit triggers an event.
0b1	A 1 to 0 transition of the trigger bit triggers an event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

Enables the generation of an event stream from CNTPCT_EL0 as seen from EL2.

EVNTEN	Meaning
0b0	Disables the event stream.
0b1	Enables the event stream.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL1PCEN, bit [1]

Traps EL0 and EL1 accesses to the EL1 physical timer registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to CNTP_CTL_EL0, CNTP_CVAL_EL0, CNTP_TVAL_EL0 are trapped to EL2, reported using EC syndrome value 0x18, unless they are trapped by CNTKCTL_EL1.EL0PTEN.
- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2 and reported using EC syndrome value 0x03, and MRRC and MCRR accesses are trapped to EL2, reported using EC syndrome value 0x04, unless they are trapped by CNTKCTL_EL1.EL0PTEN or CNTKCTL.PL0PTEN:
    - CNTP_CTL, CNTP_CVAL, CNTP_TVAL.

EL1PCEN	Meaning
0b0	EL0 and EL1 accesses to the specified registers are trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL1PCTEN, bit [0]

Traps EL0 and EL1 accesses to the EL1 physical counter registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to CNTPCT_EL0 and CNTPCTSS_EL0 are trapped to EL2 and reported using EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0PCTEN.
- In AArch32 state, MRRC or MCRR accesses to CNTPCT are trapped to EL2 and reported using EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0PCTEN or CNTKCTL.PL0PCTEN.

EL1PCTEN	Meaning
`0b0`	EL0 and EL1 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHCTL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name CNTHCTL_EL2 or CNTKCTL_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHCTL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTHCTL_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTHCTL_EL2;
```

```
MSR CNTHCTL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHCTL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTHCTL_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTKCTL_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    X[t, 64] = CNTKCTL_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTHCTL_EL2_VHE(CNTHCTL_EL2);
    else
        X[t, 64] = CNTKCTL_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTKCTL_EL1;
```

When FEAT_VHE is implemented

```
MSR CNTKCTL_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTHCTL_EL2 = CNTHCTL_EL2_VHE(X[t, 64]);
    else
        CNTKCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTKCTL_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHP_CTL_EL2, Counter-timer Hypervisor Physical Timer Control Register

The CNTHP_CTL_EL2 characteristics are:

Purpose

Control register for the EL2 physical timer.

Configuration

AArch64 System register CNTHP_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHP_CTL[31:0].

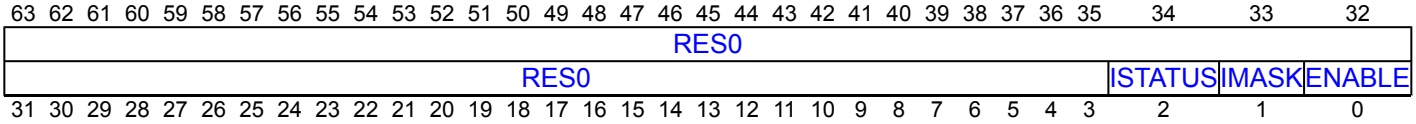
This register is present only when (EL3 is implemented or (EL3 is not implemented, EL2 is implemented, and FEAT_SEL2 is not implemented)) and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHP_CTL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHP_CTL_EL2 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHP_TVAL_EL2 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_CTL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name CNTHP_CTL_EL2 or CNTP_CTL_EL0 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHP_CTL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b001

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTHP_CTL_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTHP_CTL_EL2;
```

```
MSR CNTHP_CTL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b001

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CTL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTHP_CTL_EL2 = X[t, 64];
```

MRS <Xt>, CNTP_CTL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHPS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CTL_EL2;
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x180];
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CTL_EL2;
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CTL_EL0;
```

MSR CNTP_CTL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x180] = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CTL_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHP_CVAL_EL2, Counter-timer Physical Timer CompareValue Register (EL2)

The CNTHP_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the EL2 physical timer.

Configuration

AArch64 System register CNTHP_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTHP_CVAL[63:0].

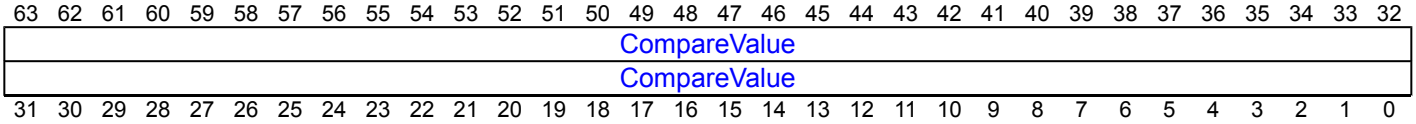
This register is present only when (EL3 is implemented or (EL3 is not implemented, EL2 is implemented, and FEAT_SEL2 is not implemented)) and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHP_CVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHP_CVAL_EL2 is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When CNTHP_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL2 physical timer, the timer condition is met and all of the following are true:

- CNTHP_CTL_EL2.ISTATUS is set to 1.
- If CNTHP_CTL_EL2.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTHP_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_CVAL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name CNTHP_CVAL_EL2 or CNTP_CVAL_EL0 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHP_CVAL_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b010

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTHP_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTHP_CVAL_EL2;
```

MSR CNTHP_CVAL_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b010

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTHP_CVAL_EL2 = X[t, 64];
```

MRS <Xt>, CNTP_CVAL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x178];
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CVAL_EL0;
```

```
MSR CNTP_CVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x178] = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHP_TVAL_EL2, Counter-timer Physical Timer TimerValue Register (EL2)

The CNTHP_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the EL2 physical timer.

Configuration

AArch64 System register CNTHP_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHP_TVAL[31:0].

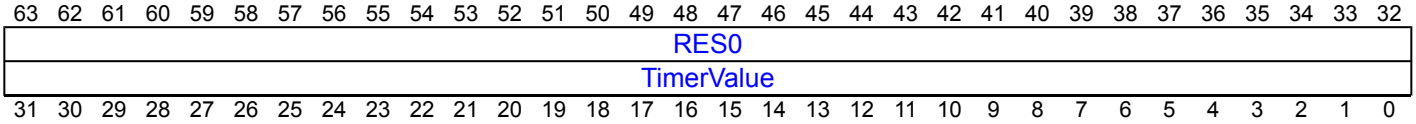
This register is present only when (EL3 is implemented or (EL3 is not implemented, EL2 is implemented, and FEAT_SEL2 is not implemented)) and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHP_TVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHP_TVAL_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If CNTHP_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHP_CTL_EL2.ENABLE is 1, the value returned is (CNTHP_CVAL_EL2 - CNTPCT_EL0).

On a write of this register, CNTHP_CVAL_EL2 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHP_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTHP_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHP_CTL_EL2.ISTATUS is set to 1.
- If CNTHP_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHP_CTL_EL2.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_TVAL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name CNTHP_TVAL_EL2 or CNTP_TVAL_EL0 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHP_TVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b000

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if CNTHP_CTL_EL2.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTHP_CTL_EL2.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTHP_TVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b000

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

```
MRS <Xt>, CNTP_TVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTP_CTL_EL0.ENABLE == '0' then
```

```
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

MSR CNTP_TVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHPS_CTL_EL2, Counter-timer Secure Physical Timer Control Register (EL2)

The CNTHPS_CTL_EL2 characteristics are:

Purpose

Control register for the Secure EL2 physical timer.

Configuration

AArch64 System register CNTHPS_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHPS_CTL[31:0].

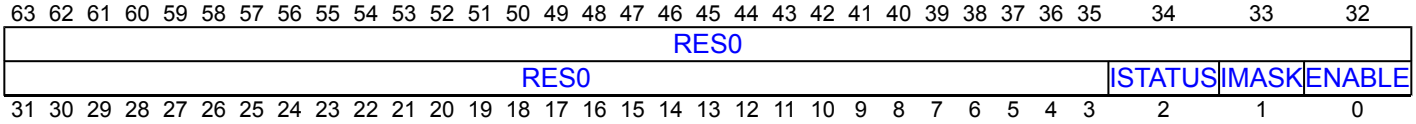
This register is present only when FEAT_SEL2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHPS_CTL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHPS_CTL_EL2 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the CNTHPS_CTL_EL2.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the CNTHPS_CTL_EL2.ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHPS_TVAL_EL2 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_CTL_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHPS_CTL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = CNTHPS_CTL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = CNTHPS_CTL_EL2;
```

```
MSR CNTHPS_CTL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2

0b11	0b100	0b1110	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHPS_CTL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CTL_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTP_CTL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHPS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CTL_EL2;
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x180];
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CTL_EL2;
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CTL_EL0;
```

When FEAT_VHE is implemented

```
MSR CNTP_CTL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x180] = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CTL_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHPS_CVAL_EL2, Counter-timer Secure Physical Timer CompareValue Register (EL2)

The CNTHPS_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the Secure EL2 physical timer.

Configuration

AArch64 System register CNTHPS_CVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHPS_CVAL[31:0].

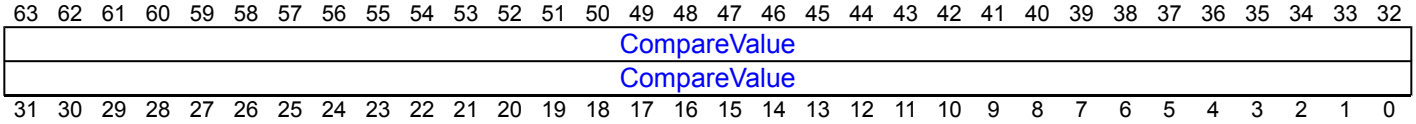
This register is present only when EL2 is implemented, FEAT_SEL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHPS_CVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHPS_CVAL_EL2 is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When CNTHPS_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the Secure EL2 physical timer, the timer condition is met and all of the following are true:

- CNTHPS_CTL_EL2.ISTATUS is set to 1.
- If CNTHPS_CTL_EL2.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTHPS_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_CVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHPS_CVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b010

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = CNTHPS_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = CNTHPS_CVAL_EL2;
```

```
MSR CNTHPS_CVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b010

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTP_CVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x178];
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CVAL_EL0;
```

When FEAT_VHE is implemented

```
MSR CNTP_CVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x178] = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHPS_TVAL_EL2, Counter-timer Secure Physical Timer TimerValue Register (EL2)

The CNTHPS_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the Secure EL2 physical timer.

Configuration

AArch64 System register CNTHPS_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHPS_TVAL[31:0].

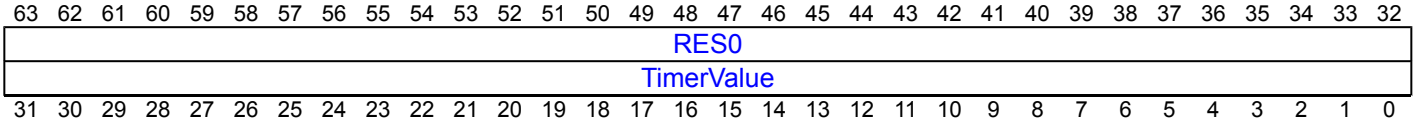
This register is present only when EL2 is implemented, FEAT_SEL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHPS_TVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHPS_TVAL_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If CNTHPS_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHPS_CTL_EL2.ENABLE is 1, the value returned is (CNTHPS_CVAL_EL2 - CNTPCT_EL0).

On a write of this register, CNTHPS_CVAL_EL2 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHPS_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTHPS_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHPS_CTL_EL2.ISTATUS is set to 1.
- If CNTHPS_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHPS_CTL_EL2.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_TVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHPS_TVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b000

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTHPS_TVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b000

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTP_TVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTP_CTL_EL0.ENABLE == '0' then
```

```
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

When FEAT_VHE is implemented

```
MSR CNTP_TVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_CTL_EL2, Counter-timer Virtual Timer Control Register (EL2)

The CNTHV_CTL_EL2 characteristics are:

Purpose

Control register for the EL2 virtual timer.

Configuration

AArch64 System register CNTHV_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHV_CTL[31:0].

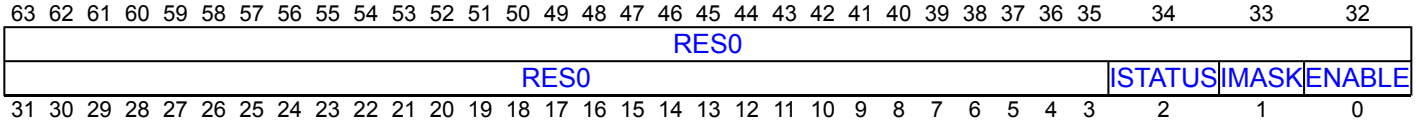
This register is present only when FEAT_VHE is implemented, (EL3 is implemented or (EL3 is not implemented and FEAT_SEL2 is not implemented)), and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHV_CTL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHV_CTL_EL2 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHV_TVAL_EL2 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHV_CTL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name CNTHV_CTL_EL2 or CNTV_CTL_EL0 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHV_CTL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTHV_CTL_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTHV_CTL_EL2;
```

```
MSR CNTHV_CTL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_CTL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTHV_CTL_EL2 = X[t, 64];
```

MRS <Xt>, CNTV_CTL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x170];
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CTL_EL0;
```

MSR CNTV_CTL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x170] = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CTL_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_CVAL_EL2, Counter-timer Virtual Timer CompareValue Register (EL2)

The CNTHV_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the EL2 virtual timer.

Configuration

AArch64 System register CNTHV_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTHV_CVAL[63:0].

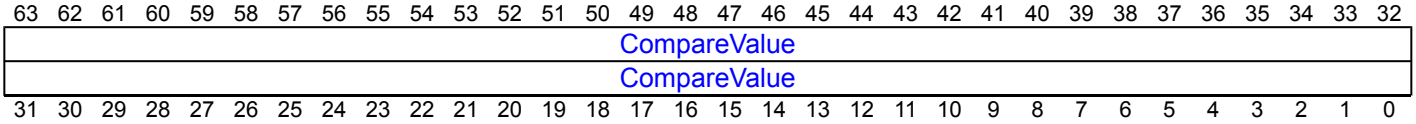
This register is present only when FEAT_VHE is implemented, (EL3 is implemented or (EL3 is not implemented and FEAT_SEL2 is not implemented)), and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHV_CVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHV_CVAL_EL2 is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the EL2 virtual timer CompareValue.

When CNTHV_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL2 virtual timer, the timer condition is met and all of the following are true:

- CNTHV_CTL_EL2.ISTATUS is set to 1.
- If CNTHV_CTL_EL2.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTHV_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHV_CVAL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name CNTHV_CVAL_EL2 or CNTV_CVAL_EL0 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHV_CVAL_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTHV_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTHV_CVAL_EL2;
```

MSR CNTHV_CVAL_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_CVAL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTHV_CVAL_EL2 = X[t, 64];
```

MRS <Xt>, CNTV_CVAL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x168];
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CVAL_EL0;
```

MSR CNTV_CVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x168] = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_TVAL_EL2, Counter-timer Virtual Timer TimerValue Register (EL2)

The CNTHV_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the EL2 virtual timer.

Configuration

AArch64 System register CNTHV_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHV_TVAL[31:0].

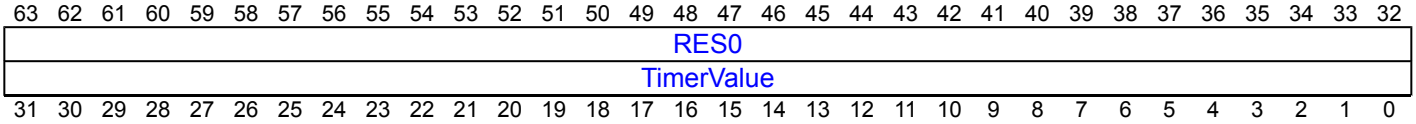
This register is present only when FEAT_VHE is implemented, (EL3 is implemented or (EL3 is not implemented and FEAT_SEL2 is not implemented)), and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHV_TVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHV_TVAL_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If CNTHV_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHV_CTL_EL2.ENABLE is 1, the value returned is (CNTHV_CVAL_EL2 - CNTVCT_EL0).

On a write of this register, CNTHV_CVAL_EL2 is set to (CNTVCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHV_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTVCT_EL0 - CNTHV_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHV_CTL_EL2.ISTATUS is set to 1.
- If CNTHV_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHV_CTL_EL2.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHV_TVAL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name CNTHV_TVAL_EL2 or CNTV_TVAL_EL0 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHV_TVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if CNTHV_CTL_EL2.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTHV_CTL_EL2.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTHV_TVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

```
MRS <Xt>, CNTV_TVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif !ELIsInHost(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTV_CTL_EL0.ENABLE == '0' then
```

```
        X[t, 64] = bits(64) UNKNOWN;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF))<31:0>, 64);
    else
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

MSR CNTV_TVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif !ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHVS_CTL_EL2, Counter-timer Secure Virtual Timer Control Register (EL2)

The CNTHVS_CTL_EL2 characteristics are:

Purpose

Control register for the Secure EL2 virtual timer.

Configuration

AArch64 System register CNTHVS_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHVS_CTL[31:0].

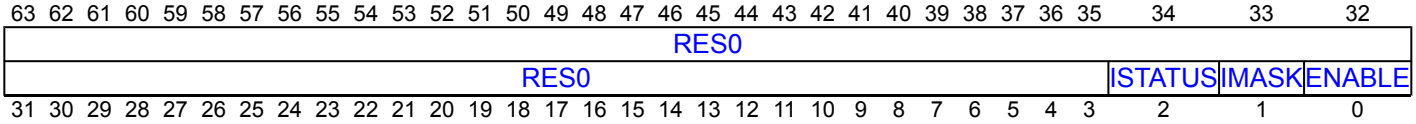
This register is present only when FEAT_SEL2 is implemented, FEAT_VHE is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHVS_CTL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHVS_CTL_EL2 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the CNTHVS_CTL_EL2.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the CNTHVS_CTL_EL2.ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHVS_TVAL_EL2 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_CTL_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHVS_CTL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = CNTHVS_CTL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = CNTHVS_CTL_EL2;
```

```
MSR CNTHVS_CTL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHVS_CTL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHVS_CTL_EL2 = X[t, 64];
```

```
MRS <Xt>, CNTV_CTL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x170];
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CTL_EL0;
```

```
MSR CNTV_CTL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x170] = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CTL_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHVS_CVAL_EL2, Counter-timer Secure Virtual Timer CompareValue Register (EL2)

The CNTHVS_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the Secure EL2 virtual timer.

Configuration

AArch64 System register CNTHVS_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTHVS_CVAL[63:0].

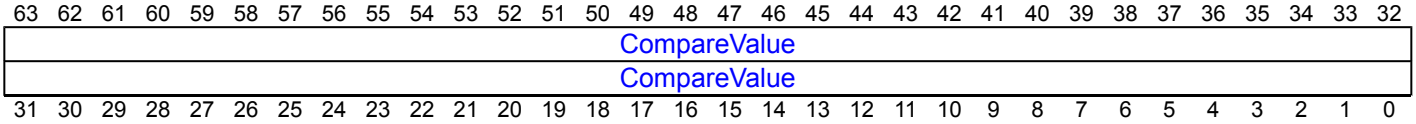
This register is present only when FEAT_SEL2 is implemented, FEAT_VHE is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHVS_CVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHVS_CVAL_EL2 is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the Secure EL2 virtual timer CompareValue.

When CNTHVS_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the Secure EL2 virtual timer, the timer condition is met and all of the following are true:

- CNTHVS_CTL_EL2.ISTATUS is set to 1.
- If CNTHVS_CTL_EL2.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTHVS_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_CVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHVS_CVAL_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = CNTHVS_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = CNTHVS_CVAL_EL2;
```

MSR CNTHVS_CVAL_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = X[t, 64];
```

MRS <Xt>, CNTV_CVAL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x168];
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CVAL_EL0;
```

MSR CNTV_CVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x168] = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHVS_TVAL_EL2, Counter-timer Secure Virtual Timer TimerValue Register (EL2)

The CNTHVS_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the Secure EL2 virtual timer.

Configuration

AArch64 System register CNTHVS_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHVS_TVAL[31:0].

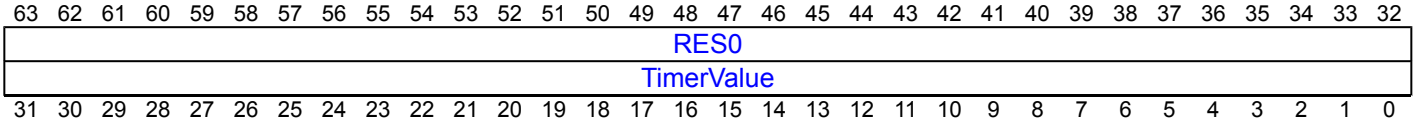
This register is present only when FEAT_SEL2 is implemented, FEAT_VHE is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHVS_TVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHVS_TVAL_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If CNTHVS_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHVS_CTL_EL2.ENABLE is 1, the value returned is (CNTHVS_CVAL_EL2 - CNTVCT_EL0).

On a write of this register, CNTHVS_CVAL_EL2 is set to (CNTVCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHVS_CTL_EL2.ENABLE is 1, the timer condition is met when ((CNTVCT_EL0 - CNTHVS_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHVS_CTL_EL2.ISTATUS is set to 1.
- If CNTHVS_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHVS_CTL_EL2.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_TVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHVS_TVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTHVS_TVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

MRS <Xt>, CNTV_TVAL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif !ELIsInHost(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTV_CTL_EL0.ENABLE == '0' then
```

```
        X[t, 64] = bits(64) UNKNOWN;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF))<31:0>, 64);
    else
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

MSR CNTV_TVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif !ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTKCTL_EL1, Counter-timer Kernel Control Register

The CNTKCTL_EL1 characteristics are:

Purpose

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this register does not cause any event stream from the virtual counter to be generated, and does not control access to the counters and timers. The access to counters and timers at EL0 is controlled by CNTHCTL_EL2.

When FEAT_VHE is not implemented, or when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, this register controls the generation of an event stream from the virtual counter, and access from EL0 to the physical counter, virtual counter, EL1 physical timers, and the virtual timer.

Configuration

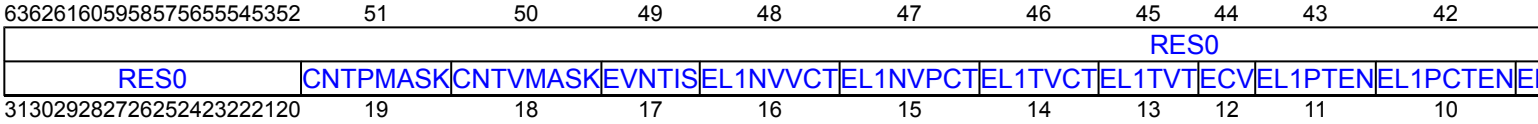
AArch64 System register CNTKCTL_EL1 bits [31:0] are architecturally mapped to AArch32 System register CNTKCTL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTKCTL_EL1 are UNDEFINED.

Attributes

CNTKCTL_EL1 is a 64-bit register.

Field descriptions



Bits [63:20]

Reserved, RES0.

CNTPMASK, bit [19]

When FEAT_RME is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CNTVMASK, bit [18]

When FEAT_RME is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVNTIS, bit [17]
When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

EVNTIS	Meaning
0b0	The CNTKCTL_EL1.EVNTI field applies to CNTVCT_EL0[15:0].
0b1	The CNTKCTL_EL1.EVNTI field applies to CNTVCT_EL0[23:8].

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVVCT, bit [16]
When FEAT_ECV is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVPCT, bit [15]
When FEAT_ECV is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVCT, bit [14]
When FEAT_ECV is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVT, bit [13]
When FEAT_ECV is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECV, bit [12]
When FEAT_ECV is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1PTEN, bit [11]
When FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1PCTEN, bit [10]
When FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL0PTEN, bit [9]

Traps EL0 accesses to the physical timer registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, the following registers are trapped and reported using EC syndrome value `0x18`:
  - CNTP_CTL_EL0, CNTP_CVAL_EL0, and CNTP_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03`, MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`:
  - CNTP_CTL, CNTP_CVAL, CNTP_TVAL.

EL0PTEN	Meaning
`0b0`	EL0 accesses to the physical timer registers are trapped to EL1.
`0b1`	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0VTEN, bit [8]

Traps EL0 accesses to the virtual timer registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped and reported using EC syndrome value `0x18`:
  - CNTV_CTL_EL0, CNTV_CVAL_EL0, and CNTV_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03`, MRRC and MCRR accesses are trapped using EC syndrome value `0x04`:
  - CNTV_CTL, CNTV_CVAL, and CNTV_TVAL.

EL0VTEN	Meaning
`0b0`	EL0 accesses to the virtual timer registers are trapped.
`0b1`	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTI, bits [7:4]

Selects which bit of CNTVCT_EL0, as seen from EL1, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTKCTL_EL1.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of CNTVCT_EL0.

Otherwise, this field selects a trigger bit in the range 0 to 15 of CNTVCT_EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the CNTVCT_EL0 trigger bit, as seen from EL1 and defined by EVNTI, generates an event when the event stream is enabled.

EVNTDIR	Meaning
0b0	A 0 to 1 transition of the trigger bit triggers an event.
0b1	A 1 to 0 transition of the trigger bit triggers an event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

When FEAT_VHE is not implemented, or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, enables the generation of an event stream from CNTVCT_EL0 as seen from EL1.

EVNTEN	Meaning
0b0	Disables the event stream.
0b1	Enables the event stream.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not enable the event stream.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0VCTEN, bit [1]

Traps EL0 accesses to the frequency register and virtual counter registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped and reported using EC syndrome value 0x18:
  - CNTVCT_EL0.
  - CNTVCTSS_EL0.
  - If CNTKCTL_EL1.EL0PCTEN is 0, CNTFRQ_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value 0x03, MRRC and MCRR accesses are trapped and reported using EC syndrome value 0x04:
  - CNTVCT and if CNTKCTL_EL1.EL0PCTEN is 0, CNTFRQ.

EL0VCTEN	Meaning
0b0	EL0 accesses to the frequency register and virtual counter registers are trapped.
0b1	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0PCTEN, bit [0]

Traps EL0 accesses to the frequency register and physical counter register to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, the following registers are trapped and reported using EC syndrome value 0x18:
  - CNTPCT_EL0
  - CNTPCTSS_EL0
  - If CNTKCTL_EL1.EL0VCTEN is 0, CNTFRQ_EL0.
- In AArch32 state, MCR or MRC accesses the following registers are trapped and reported using EC syndrome value 0x03, MCRR or MRRC accesses are trapped and reported using EC syndrome value 0x04:
  - CNTPCT and if CNTKCTL_EL1.EL0VCTEN is 0, CNTFRQ.

EL0PCTEN	Meaning
0b0	EL0 accesses to the frequency register and physical counter register are trapped.
0b1	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTKCTL_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name CNTKCTL_EL1 or CNTKCTL_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTKCTL_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    X[t, 64] = CNTKCTL_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTHCTL_EL2_VHE(CNTHCTL_EL2);
    else
        X[t, 64] = CNTKCTL_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTKCTL_EL1;
```

```
MSR CNTKCTL_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTHCTL_EL2 = CNTHCTL_EL2_VHE(X[t, 64]);
    else
        CNTKCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTKCTL_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTKCTL_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTKCTL_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTKCTL_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR CNTKCTL_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTKCTL_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTKCTL_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_CTL_EL0, Counter-timer Physical Timer Control Register

The CNTP_CTL_EL0 characteristics are:

Purpose

Control register for the EL1 physical timer.

Configuration

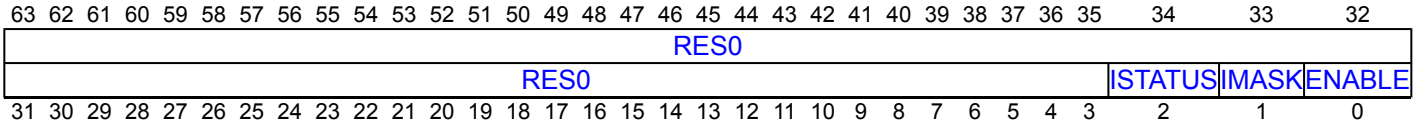
AArch64 System register CNTP_CTL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTP_CTL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTP_CTL_EL0 are UNDEFINED.

Attributes

CNTP_CTL_EL0 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTP_TVAL_EL0 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CTL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name CNTP_CTL_EL0 or CNTP_CTL_EL02 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTP_CTL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHPS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CTL_EL2;
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x180];
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CTL_EL2;
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CTL_EL0;
```

MSR CNTP_CTL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x180] = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CTL_EL0 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTP_CTL_EL02
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            X[t, 64] = NVMem[0x180];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTP_CTL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTP_CTL_EL0;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR CNTP_CTL_EL02, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x180] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTP_CTL_EL0 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTP_CTL_EL0 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_CVAL_EL0, Counter-timer Physical Timer CompareValue Register

The CNTP_CVAL_EL0 characteristics are:

Purpose

Holds the compare value for the EL1 physical timer.

Configuration

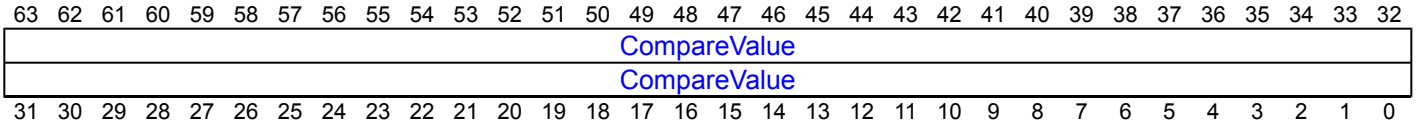
AArch64 System register CNTP_CVAL_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTP_CVAL[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTP_CVAL_EL0 are UNDEFINED.

Attributes

CNTP_CVAL_EL0 is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the EL1 physical timer CompareValue.

When CNTP_CTL_EL0.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 physical timer, the timer condition is met and all of the following are true:

- CNTP_CTL_EL0.ISTATUS is set to 1.
- If CNTP_CTL_EL0.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTP_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CVAL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name CNTP_CVAL_EL0 or CNTP_CVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTP_CVAL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x178];
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CVAL_EL0;
```

MSR CNTP_CVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x178] = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTP_CVAL_EL02
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            X[t, 64] = NVMem[0x178];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTP_CVAL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTP_CVAL_EL0;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR CNTP_CVAL_EL02, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x178] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTP_CVAL_EL0 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTP_CVAL_EL0 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_TVAL_EL0, Counter-timer Physical Timer TimerValue Register

The CNTP_TVAL_EL0 characteristics are:

Purpose

Holds the timer value for the EL1 physical timer.

Configuration

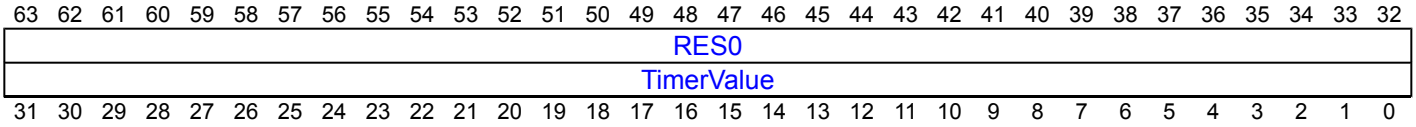
AArch64 System register CNTP_TVAL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTP_TVAL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTP_TVAL_EL0 are UNDEFINED.

Attributes

CNTP_TVAL_EL0 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If CNTP_CTL_EL0.ENABLE is 0, the value returned is UNKNOWN.
- If CNTP_CTL_EL0.ENABLE is 1, the value returned is (CNTP_CVAL_EL0 - CNTPCT_EL0).

On a write of this register, CNTP_CVAL_EL0 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTP_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTP_CVAL_EL0) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTP_CTL_EL0.ISTATUS is set to 1.
- If CNTP_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTP_CTL_EL0.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

Note

The value of CNTPCT_EL0 used in these calculations is the value seen at the Exception level that the CNTPCT_EL0 register is being read or written from.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_TVAL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name CNTP_TVAL_EL0 or CNTP_TVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTP_TVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTP_CTL_EL0.ENABLE == '0' then
```

```
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTP_TVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTP_TVAL_EL02
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR CNTP_TVAL_EL02, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPCT_EL0, Counter-timer Physical Count Register

The CNTPCT_EL0 characteristics are:

Purpose

Reads of CNTPCT_EL0 return the 64-bit physical count value minus a physical offset.

Configuration

AArch64 System register CNTPCT_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTPCT[63:0].

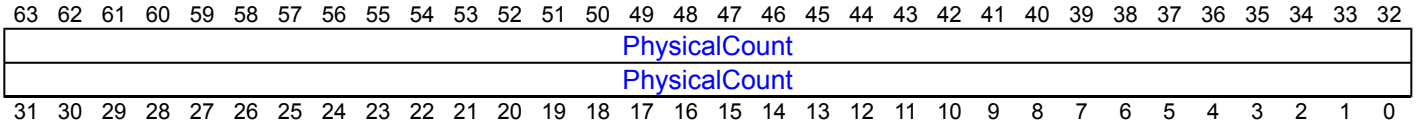
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPCT_EL0 are UNDEFINED.

All reads to the CNTPCT_EL0 occur in program order relative to reads to CNTPCTSS_EL0 or CNTPCT_EL0.

Attributes

CNTPCT_EL0 is a 64-bit register.

Field descriptions



PhysicalCount, bits [63:0]

Physical count value.

If the access is not trapped and all of the following are true, then reads of CNTPCT_EL0 from EL0 or EL1 return (PhysicalCountInt<63:0> - CNTPOFF_EL2<63:0>):

- EL2 is implemented and enabled in the current Security state.
- CNTHCTL_EL2.ECV is 1.
- SCR_EL3.ECVEn is 1.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise, reads of CNTPCT_EL0 return PhysicalCountInt<63:0>.

PhysicalCountInt is defined by 'The Physical Counter'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCT_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTPCT_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn
== '1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
            X[t, 64] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn
== '1') && CNTHCTL_EL2.ECV == '1' then
            X[t, 64] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    X[t, 64] = PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPCTSS_EL0, Counter-timer Self-Synchronized Physical Count Register

The CNTPCTSS_EL0 characteristics are:

Purpose

Holds the self-synchronized view of the 64-bit physical count value.

Configuration

AArch64 System register CNTPCTSS_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTPCTSS[63:0].

This register is present only when FEAT_ECV is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPCTSS_EL0 are UNDEFINED.

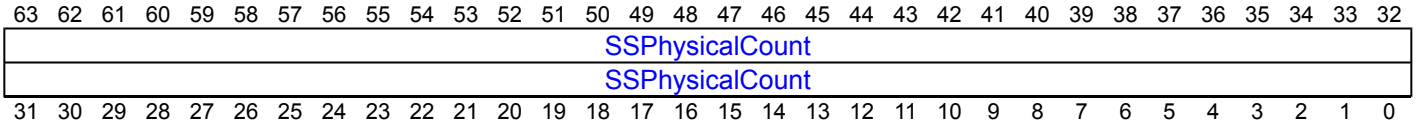
All reads to the CNTPCTSS_EL0 occur in program order relative to reads to CNTPCT_EL0 or CNTPCTSS_EL0.

This register is a view of the CNTPCT_EL0 register for which reads appear to occur in program order relative to other instructions, without the need for any explicit synchronization. Reads of this register return a value consistent with the counter not being read until the read instruction is known to be non-speculative.

Attributes

CNTPCTSS_EL0 is a 64-bit register.

Field descriptions



Physical count value.

If the access is not trapped and all of the following are true, then reads of CNTPCTSS_EL0 from EL0 or EL1 return (PhysicalCountInt<63:0> - CNTPOFF_EL2<63:0>):

- EL2 is implemented and enabled in the current Security state.
- CNTHCTL_EL2.ECV is 1.
- SCR_EL3.ECVEn is 1.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise, reads of CNTPCTSS_EL0 return PhysicalCountInt<63:0>.

PhysicalCountInt is defined by 'The Physical Counter'.

SSPhysicalCount, bits [63:0]

Self-synchronized physical count value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCTSS_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTPCTSS_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_ECV) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn
== '1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
            X[t, 64] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn
== '1') && CNTHCTL_EL2.ECV == '1' then
            X[t, 64] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    X[t, 64] = PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPOFF_EL2, Counter-timer Physical Offset Register

The CNTPOFF_EL2 characteristics are:

Purpose

Holds the 64-bit physical offset. This is the offset for the AArch64 EL1 physical timer and counter when Enhanced Counter Virtualization is enabled.

Configuration

This register is present only when FEAT_ECV_POFF is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPOFF_EL2 are UNDEFINED.

The physical offset applies to:

- Direct reads of CNTPCT_EL0 from EL0 or EL1.
- Indirect reads of the physical counter by the EL1 physical timer. See 'The Generic Timer in AArch64 state'.
- Indirect reads of the physical counter for timestamps generated by profiling logic. See 'The Statistical Profiling Extension' and 'AArch64 Self-hosted Trace'.

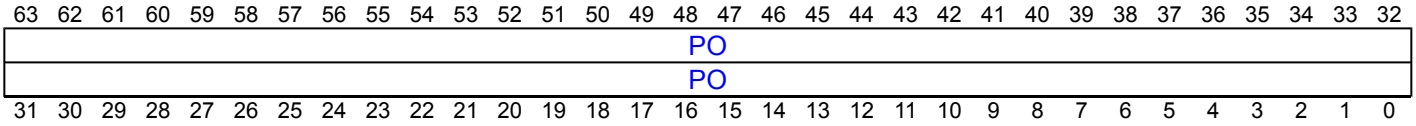
The physical offset only applies under conditions described by the relevant sections.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTPOFF_EL2 is a 64-bit register.

Field descriptions



PO, bits [63:0]

Physical offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPOFF_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTPOFF_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_ECV_POFF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1A8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ECVEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ECVEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = CNTPOFF_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTPOFF_EL2;
```

MSR CNTPOFF_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_ECV_POFF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1A8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ECVEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ECVEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        CNTPOFF_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTPOFF_EL2 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPS_CTL_EL1, Counter-timer Physical Secure Timer Control Register

The CNTPS_CTL_EL1 characteristics are:

Purpose

Control register for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

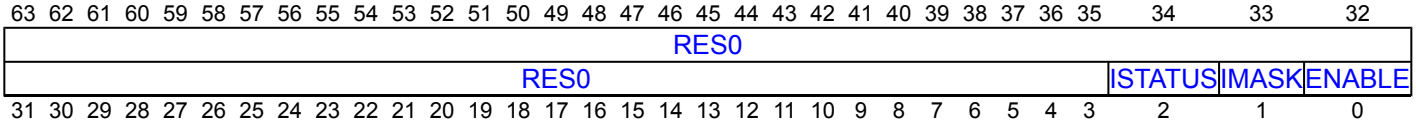
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPS_CTL_EL1 are UNDEFINED.

Attributes

CNTPS_CTL_EL1 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTPS_TVAL_EL1 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPS_CTL_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTPS_CTL_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = CNTPS_CTL_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTPS_CTL_EL1;
```

```
MSR CNTPS_CTL_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CNTPS_CTL_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    CNTPS_CTL_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPS_CVAL_EL1, Counter-timer Physical Secure Timer CompareValue Register

The CNTPS_CVAL_EL1 characteristics are:

Purpose

Holds the compare value for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

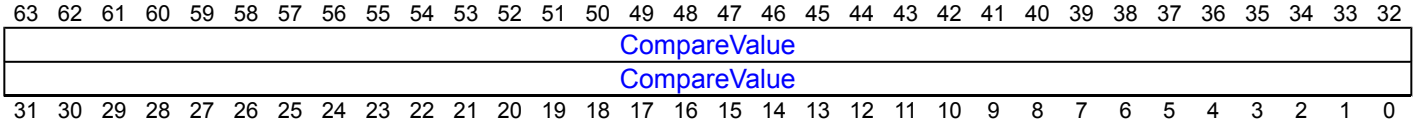
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPS_CVAL_EL1 are UNDEFINED.

Attributes

CNTPS_CVAL_EL1 is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the secure physical timer CompareValue.

When CNTPS_CTL_EL1.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 secure physical timer, the timer condition is met and all of the following are true:

- CNTPS_CTL_EL1.ISTATUS is set to 1.
- If CNTPS_CTL_EL1.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTPS_CTL_EL1.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPS_CVAL_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTPS_CVAL_EL1

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = CNTPS_CVAL_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTPS_CVAL_EL1;
```

MSR CNTPS_CVAL_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CNTPS_CVAL_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    CNTPS_CVAL_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPS_TVAL_EL1, Counter-timer Physical Secure Timer TimerValue Register

The CNTPS_TVAL_EL1 characteristics are:

Purpose

Holds the timer value for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

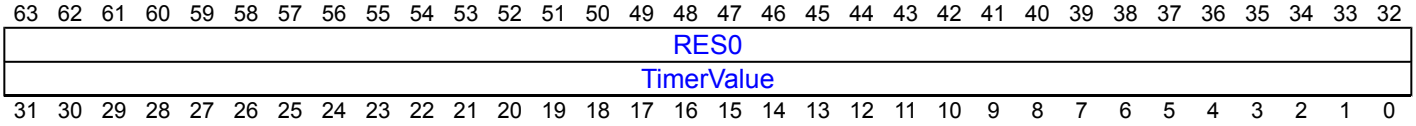
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPS_TVAL_EL1 are UNDEFINED.

Attributes

CNTPS_TVAL_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the secure physical timer.

On a read of this register:

- If CNTPS_CTL_EL1.ENABLE is 0, the value returned is UNKNOWN.
- If CNTPS_CTL_EL1.ENABLE is 1, the value returned is (CNTPS_CVAL_EL1 - CNTPCT_EL0).

On a write of this register, CNTPS_CVAL_EL1 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTPS_CTL_EL1.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTPS_CVAL_EL1) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTPS_CTL_EL1.ISTATUS is set to 1.
- If CNTPS_CTL_EL1.IMASK is 0, an interrupt is generated.

When CNTPS_CTL_EL1.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPS_TVAL_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTPS_TVAL_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            if CNTPS_CTL_EL1.ENABLE == '0' then
                X[t, 64] = bits(64) UNKNOWN;
            else
                X[t, 64] = ZeroExtend((CNTPS_CVAL_EL1 - PhysicalCountInt())<31:0>, 64);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CNTPS_CTL_EL1.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTPS_CVAL_EL1 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTPS_TVAL_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CNTPS_CVAL_EL1 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    CNTPS_CVAL_EL1 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_CTL_EL0, Counter-timer Virtual Timer Control Register

The CNTV_CTL_EL0 characteristics are:

Purpose

Control register for the virtual timer.

Configuration

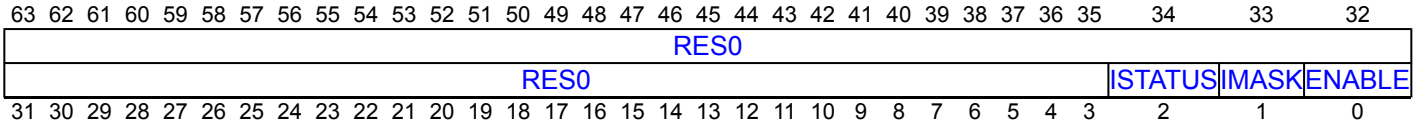
AArch64 System register CNTV_CTL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTV_CTL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTV_CTL_EL0 are UNDEFINED.

Attributes

CNTV_CTL_EL0 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTV_TVAL_EL0 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CTL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name CNTV_CTL_EL0 or CNTV_CTL_EL02 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTV_CTL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x170];
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CTL_EL0;
```

```
MSR CNTV_CTL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x170] = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CTL_EL0 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTV_CTL_EL02
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            X[t, 64] = NVMem[0x170];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTV_CTL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTV_CTL_EL0;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR CNTV_CTL_EL02, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x170] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTV_CTL_EL0 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTV_CTL_EL0 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_CVAL_EL0, Counter-timer Virtual Timer CompareValue Register

The CNTV_CVAL_EL0 characteristics are:

Purpose

Holds the compare value for the virtual timer.

Configuration

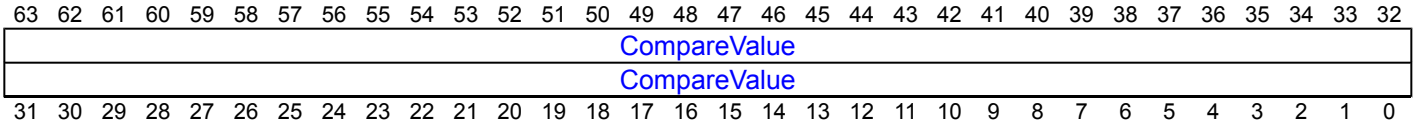
AArch64 System register CNTV_CVAL_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTV_CVAL[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTV_CVAL_EL0 are UNDEFINED.

Attributes

CNTV_CVAL_EL0 is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the EL1 virtual timer CompareValue.

When CNTV_CTL_EL0.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 virtual timer, the timer condition is met and all of the following are true:

- CNTV_CTL_EL0.ISTATUS is set to 1.
- If CNTV_CTL_EL0.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTV_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CVAL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name CNTV_CVAL_EL0 or CNTV_CVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTV_CVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x168];
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CVAL_EL0;
```

```
MSR CNTV_CVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x168] = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL_EL0 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTV_CVAL_EL02
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            X[t, 64] = NVMem[0x168];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTV_CVAL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTV_CVAL_EL0;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR CNTV_CVAL_EL02, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x168] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_TVAL_EL0, Counter-timer Virtual Timer TimerValue Register

The CNTV_TVAL_EL0 characteristics are:

Purpose

Holds the timer value for the EL1 virtual timer.

Configuration

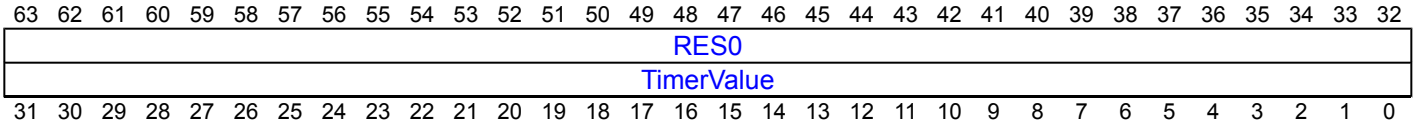
AArch64 System register CNTV_TVAL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTV_TVAL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTV_TVAL_EL0 are UNDEFINED.

Attributes

CNTV_TVAL_EL0 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL1 virtual timer.

On a read of this register:

- If CNTV_CTL_EL0.ENABLE is 0, the value returned is UNKNOWN.
- If CNTV_CTL_EL0.ENABLE is 1, the value returned is (CNTV_CVAL_EL0 - CNTVCT_EL0).

On a write of this register, CNTV_CVAL_EL0 is set to (CNTVCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTV_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTVCT_EL0 - CNTV_CVAL_EL0) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTV_CTL_EL0.ISTATUS is set to 1.
- If CNTV_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTV_CTL_EL0.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_TVAL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name CNTV_TVAL_EL0 or CNTV_TVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTV_TVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif !ELIsInHost(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTV_CTL_EL0.ENABLE == '0' then
```

```
        X[t, 64] = bits(64) UNKNOWN;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF))<31:0>, 64);
    else
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTV_TVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif !ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

When FEAT_VHE is implemented

```
MRS <Xt>, CNTV_TVAL_EL02
```

op0	op1	CRn	CRm	op2

0b11	0b101	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR CNTV_TVAL_EL02, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVCT_EL0, Counter-timer Virtual Count Register

The CNTVCT_EL0 characteristics are:

Purpose

Reads of CNTVCT_EL0 return the 64-bit physical count value minus a virtual offset.

Configuration

AArch64 System register CNTVCT_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTVCT[63:0].

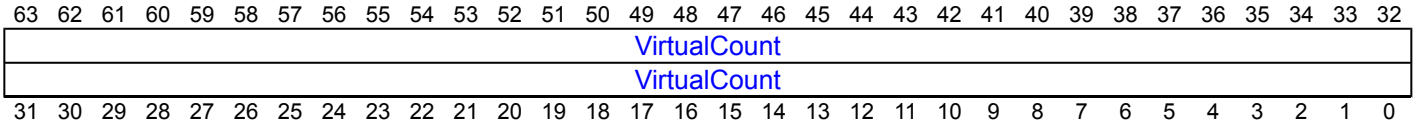
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTVCT_EL0 are UNDEFINED.

All reads to the CNTVCT_EL0 occur in program order relative to reads to CNTVCTSS_EL0 or CNTVCT_EL0.

Attributes

CNTVCT_EL0 is a 64-bit register.

Field descriptions



VirtualCount, bits [63:0]

Virtual count value.

When EL2 is implemented, if the access is not trapped, and any of the following are true, then reads of CNTVCT_EL0 from EL0 and EL1 return (PhysicalCountInt<63:0> - CNTVOFF_EL2<63:0>):

- All of the following are true:
  - The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
  - CNTVCT_EL0 is read from EL0.
- All of the following are true:
  - The Effective value of HCR_EL2.E2H is 0.
  - CNTVCT_EL0 is read from EL2.
- CNTVCT_EL0 is read from EL1 or EL3.

Otherwise reads of CNTVCT_EL0 return PhysicalCountInt<63:0>.

PhysicalCountInt is defined by 'The Physical Counter'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCT_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTVCT_EL0
```

op0	op1	CRn	CRm	op2

0b11	0b011	0b1110	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) && (!EL2Enabled() || !ELIsInHost(EL0)) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) && !ELUsingAArch32(EL2) &&
(!EL2Enabled() || !ELIsInHost(EL0)) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL2) && !ELIsInHost(EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
    else
        X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF;
    else
        X[t, 64] = PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVCTSS_EL0, Counter-timer Self-Synchronized Virtual Count Register

The CNTVCTSS_EL0 characteristics are:

Purpose

Reads of CNTVCTSS_EL0 return the 64-bit physical count value minus a virtual offset.

Configuration

AArch64 System register CNTVCTSS_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTVCTSS[63:0].

This register is present only when FEAT_ECV is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTVCTSS_EL0 are UNDEFINED.

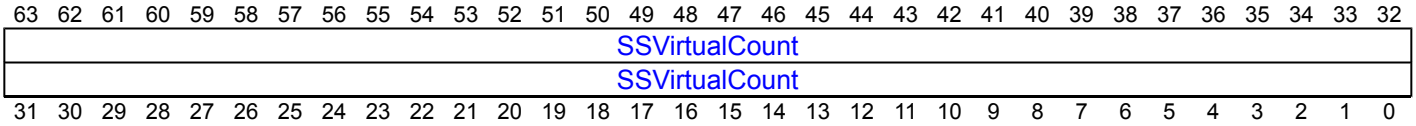
All reads to the CNTVCTSS_EL0 occur in program order relative to reads to CNTVCT_EL0 or CNTVCTSS_EL0.

This register is a view of the CNTVCT_EL0 register for which reads appear to occur in program order relative to other instructions, without the need for any explicit synchronization. Reads of this register return a value consistent with the counter not being read until the read instruction is known to be non-speculative.

Attributes

CNTVCTSS_EL0 is a 64-bit register.

Field descriptions



SSVirtualCount, bits [63:0]

Self-synchronized virtual count value.

When EL2 is implemented, if the access is not trapped, and any of the following are true, then reads of CNTVCTSS_EL0 from EL0 and EL1 return (PhysicalCountInt<63:0> - CNTVOFF_EL2<63:0>):

- All of the following are true:
  - The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
  - CNTVCTSS_EL0 is read from EL0.
- All of the following are true:
  - The Effective value of HCR_EL2.E2H is 0.
  - CNTVCTSS_EL0 is read from EL2.
- CNTVCTSS_EL0 is read from EL1 or EL3.

Otherwise reads of CNTVCTSS_EL0 return PhysicalCountInt<63:0>.

PhysicalCountInt is defined by 'The Physical Counter'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCTSS_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTVCTSS_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_ECV) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) && (!EL2Enabled() || !ELIsInHost(EL0)) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) && !ELUsingAArch32(EL2) &&
(!EL2Enabled() || !ELIsInHost(EL0)) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL2) && !ELIsInHost(EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
    else
        X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF;
    else
        X[t, 64] = PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVOFF_EL2, Counter-timer Virtual Offset Register

The CNTVOFF_EL2 characteristics are:

Purpose

Holds the 64-bit virtual offset. This is the offset for the AArch64 virtual timers and counters.

Configuration

AArch64 System register CNTVOFF_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTVOFF[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTVOFF_EL2 are UNDEFINED.

The virtual offset applies to:

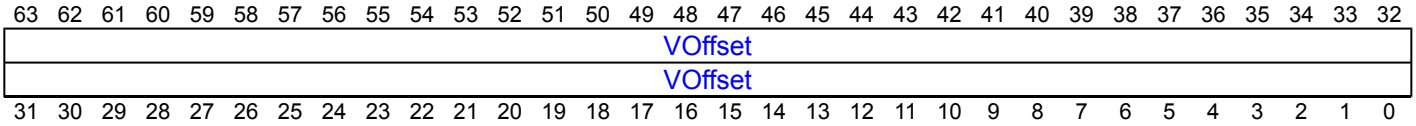
- Direct reads of CNTVCT_EL0.
- Indirect reads of the virtual counter by the EL1 virtual timer. See 'The Generic Timer in AArch64 state'.
- Indirect reads of the virtual counter for timestamps generated by profiling logic. See 'The Statistical Profiling Extension' and 'AArch64 Self-hosted Trace'.

The virtual offset only applies under conditions described by the relevant sections.

Attributes

CNTVOFF_EL2 is a 64-bit register.

Field descriptions



VOffset, bits [63:0]

Virtual offset.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVOFF_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTVOFF_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x060];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTVOFF_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTVOFF_EL2;
```

```
MSR CNTVOFF_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x060] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTVOFF_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTVOFF_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CONTEXTIDR_EL1, Context ID Register (EL1)

The CONTEXTIDR_EL1 characteristics are:

Purpose

Identifies the current Process Identifier.

The value of the whole of this register is called the Context ID and is used by:

- The debug logic, for Linked and Unlinked Context ID matching.
- The trace logic, to identify the current process.

The significance of this register is for debug and trace use only.

Configuration

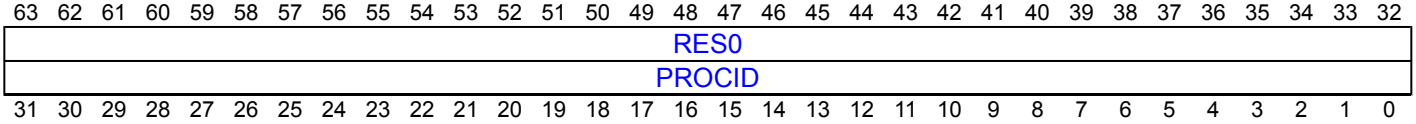
AArch64 System register CONTEXTIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CONTEXTIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CONTEXTIDR_EL1 are UNDEFINED.

Attributes

CONTEXTIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

PROCID, bits [31:0]

Process Identifier. This field must be programmed with a unique value that identifies the current process.

Note

In AArch32 state, when TTBCR.EAE is set to 0, CONTEXTIDR.ASID holds the ASID.

In AArch64 state, CONTEXTIDR_EL1 is independent of the ASID, and for the EL1&0 translation regime either TTBR0_EL1 or TTBR1_EL1 holds the ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CONTEXTIDR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name CONTEXTIDR_EL1 or CONTEXTIDR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CONTEXTIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.CONTEXTIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x108];
    else
        X[t, 64] = CONTEXTIDR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CONTEXTIDR_EL2;
    else
        X[t, 64] = CONTEXTIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CONTEXTIDR_EL1;
```

```
MSR CONTEXTIDR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.CONTEXTIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x108] = X[t, 64];
    else
        CONTEXTIDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CONTEXTIDR_EL2 = X[t, 64];
    else
        CONTEXTIDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CONTEXTIDR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CONTEXTIDR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x108];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CONTEXTIDR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CONTEXTIDR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR CONTEXTIDR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x108] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CONTEXTIDR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CONTEXTIDR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CONTEXTIDR_EL2, Context ID Register (EL2)

The CONTEXTIDR_EL2 characteristics are:

Purpose

Identifies the current Process Identifier for EL2.

The value of the whole of this register is called the Context ID and is used by:

- The debug logic, for Linked and Unlinked Context ID matching.
- The trace logic, to identify the current process.

The significance of this register is for debug and trace use only.

Configuration

This register is present only when FEAT_Debugv8p1 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CONTEXTIDR_EL2 are UNDEFINED.

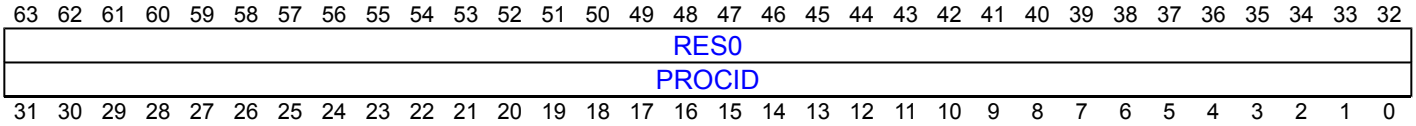
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

CONTEXTIDR_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

PROCID, bits [31:0]

Process Identifier. This field must be programmed with a unique value that identifies the current process.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CONTEXTIDR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name CONTEXTIDR_EL2 or CONTEXTIDR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CONTEXTIDR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_Debugv8p1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CONTEXTIDR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CONTEXTIDR_EL2;
```

```
MSR CONTEXTIDR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_Debugv8p1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CONTEXTIDR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CONTEXTIDR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CONTEXTIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.CONTEXTIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x108];
    else
        X[t, 64] = CONTEXTIDR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CONTEXTIDR_EL2;
    else
        X[t, 64] = CONTEXTIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CONTEXTIDR_EL1;
```

When FEAT_VHE is implemented

```
MSR CONTEXTIDR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.CONTEXTIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x108] = X[t, 64];
    else
        CONTEXTIDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CONTEXTIDR_EL2 = X[t, 64];
    else
        CONTEXTIDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CONTEXTIDR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

COSP RCTX, Clear Other Speculative Prediction Restriction by Context

The COSP RCTX characteristics are:

Purpose

Clear Other Speculative Prediction Restriction by Context applies to all prediction resources not managed by other speculation restriction System instructions.

The actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control any predictions occurring after the instruction is complete and synchronized.

This instruction applies to all speculative access except:

- Cache Prefetch predictions.
- Control Flow predictions.
- Data Value predictions.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the PE that executed the original restriction instruction, and a subsequent Context Synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of Cache Allocation Resources so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations, the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

This instruction applies to speculative accesses resulting from address predictions.

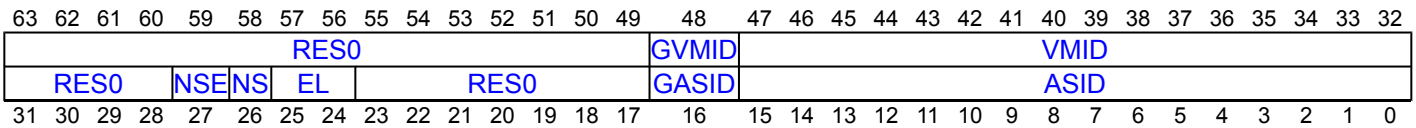
Configuration

This instruction is present only when FEAT_SPECRES2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to COSP RCTX are UNDEFINED.

Attributes

COSP RCTX is a 64-bit System instruction.

Field descriptions



Bits [63:49]

Reserved, RES0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 target execution context.
0b1	Applies to all VMIDs for an EL0 or EL1 target execution context.

For target execution contexts other than EL0 and EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and the target execution context is either:

- EL1.
- EL0 when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, this field is treated as the current VMID.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

If the implementation supports 16 bits of VMID, then the upper 8 bits of the VMID must be written to 0 by software when the context being affected only uses 8 bits.

Bits [31:28]

Reserved, RES0.

NSE, bit [27]

When FEAT_RME is implemented:

Together with the NS field, selects the Security state.

For a description of the values derived by evaluating NS and NSE together, see COSP_RCTX.NS.

Otherwise:

Reserved, RES0.

NS, bit [26]

When FEAT_RME is implemented:

Together with the NSE field, selects the Security state.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

Some Effective values are determined by the current Security state:

- When executed in Secure state, the Effective value of NSE is 0.
- When executed in Non-secure state, the Effective value of {NSE, NS} is {0, 1}.
- When executed in Realm state, the Effective value of {NSE, NS} is {1, 1}.

This instruction is treated as a NOP when executed at EL3 and either:

- COSP_RCTX.{NSE, NS} selects a reserved value.
- COSP_RCTX.{NSE, NS} == {1, 0} and COSP_RCTX.EL has a value other than `0b11`.

Otherwise:

Security State.

NS	Meaning
`0b0`	Secure state.
`0b1`	Non-secure state.

When executed in Non-secure state, the Effective value of NS is 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

EL	Meaning
`0b00`	EL0.
`0b01`	EL1.
`0b10`	EL2.
`0b11`	EL3.

If the instruction is executed at an Exception level lower than the specified level, or is specified to apply to a combination of Exception level and Security state that is not implemented, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs, or a specified ASID.

GASID	Meaning
`0b0`	Applies to specified ASID for an EL0 target execution context.
`0b1`	Applies to all ASIDs for an EL0 target execution context.

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [15:0]

Only applies to an EL0 target execution context and when bit[16] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being affected only uses 8 bits.

Executing COSP RCTX

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
COSP RCTX, <Xt>
```

op0	op1	CRn	CRm	op2

0b01	0b011	0b0111	0b0011	0b110

```
if !(IsFeatureImplemented(FEAT_SPECRES2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.COSPRCTX == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnRCTX == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.RestrictPrediction(X[t, 64], RestrictType_Other);
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.COSPRCTX == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.RestrictPrediction(X[t, 64], RestrictType_Other);
elsif PSTATE.EL == EL2 then
    AArch64.RestrictPrediction(X[t, 64], RestrictType_Other);
elsif PSTATE.EL == EL3 then
    AArch64.RestrictPrediction(X[t, 64], RestrictType_Other);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPACR_EL1, Architectural Feature Access Control Register

The CPACR_EL1 characteristics are:

Purpose

Controls access to trace, SME, Streaming SVE, SVE, and Advanced SIMD and floating-point functionality.

Configuration

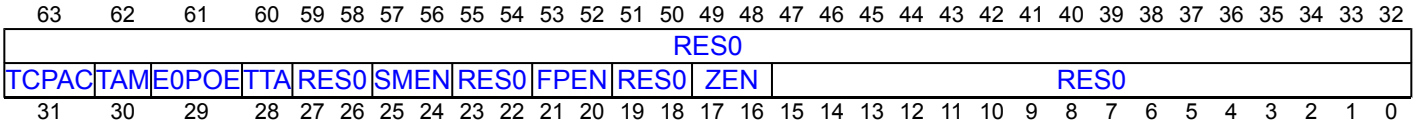
AArch64 System register CPACR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CPACR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CPACR_EL1 are UNDEFINED.

Attributes

CPACR_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

When FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TAM, bit [30]

When FEAT_AMUv1 is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [29]
When FEAT_S1POE is implemented:

Enable access to POR_EL0.

Traps EL0 accesses to POR_EL0, from AArch64 state only to EL1, or to EL2 when it is implemented and enabled in the current Security state and HCR_EL2.TGE is 1. The exception is reported using EC syndrome value 0x18.

E0POE	Meaning
0b0	This control causes EL0 access to POR_EL0 to be trapped.
0b1	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTA, bit [28]
When System register access to the trace unit registers is implemented:

Traps EL0 and EL1 System register accesses to all implemented trace registers from both Execution states to EL1, or to EL2 when it is implemented and enabled in the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to trace registers are trapped, reported using EC syndrome value 0x18.
- In AArch32 state, MRC and MCR accesses to trace registers are trapped, reported using EC syndrome value 0x05.
- In AArch32 state, MRRC and MCRR accesses to trace registers are trapped, reported using EC syndrome value 0x0C.

TTA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	This control causes EL0 and EL1 System register accesses to all implemented trace registers to be trapped.

Note

- The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPACR_EL1.TTA is 1.
- The Arm architecture does not provide traps on trace register accesses through the optional memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [27:26]

Reserved, RES0.

SMEN, bits [25:24]

When FEAT_SME is implemented:

Traps execution at EL1 and EL0 of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the SVCR or SMCR_EL1 System registers to EL1, or to EL2 when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1.

When instructions that directly access the SVCR System register are trapped with reference to this control, the `MSR SVCRSM`, `MSR SVCRZA`, and `MSR SVCRSMZA` instructions are also trapped.

The exception is reported using EC syndrome value `0x1D`, with an ISS code of `0x0000000`.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPACR_EL1.SMEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

SMEN	Meaning
`0b00`	This control causes execution of these instructions at EL1 and EL0 to be trapped.
`0b01`	This control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL1 to be trapped.
`0b10`	This control causes execution of these instructions at EL1 and EL0 to be trapped.
`0b11`	This control does not cause execution of any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [23:22]

Reserved, RES0.

FPEN, bits [21:20]

Traps execution at EL1 and EL0 of instructions that access the Advanced SIMD and floating-point registers from both Execution states to EL1, reported using EC syndrome value `0x07`, or to EL2 reported using EC syndrome value `0x00` when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to FPMR, FPCR, FPSR, and any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-31 registers.
- In AArch32 state, FPSCR, and any of the SIMD and floating-point registers Q0-15, including their views as D0-D31 registers or S0-31 registers.

Traps execution at EL1 and EL0 of SME and SVE instructions to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The exception is reported using EC syndrome value `0x07`.

A trap taken as a result of CPACR_EL1.SMEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

A trap taken as a result of CPACR_EL1.ZEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

FPEN	Meaning
0b00	This control causes execution of these instructions at EL1 and EL0 to be trapped.
0b01	This control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL1 to be trapped.
0b10	This control causes execution of these instructions at EL1 and EL0 to be trapped.
0b11	This control does not cause execution of any instructions to be trapped.

Writes to MVFR0, MVFR1, and MVFR2 from EL1 or higher are CONSTRAINED UNPREDICTABLE and whether these accesses can be trapped by this control depends on implemented CONSTRAINED UNPREDICTABLE behavior.

Note

- Attempts to write to the FPSID count as use of the registers for accesses from EL1 or higher.
- Accesses from EL0 to FPSID, MVFR0, MVFR1, MVFR2, and FPEXC are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPACR_EL1.FPEN is not 0b11.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:18]

Reserved, RES0.

ZEN, bits [17:16]

When FEAT_SVE is implemented:

Traps execution at EL1 and EL0 of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the ZCR_EL1 System register to EL1, or to EL2 when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1.

The exception is reported using EC syndrome value 0x19.

A trap taken as a result of CPACR_EL1.ZEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

ZEN	Meaning
0b00	This control causes execution of these instructions at EL1 and EL0 to be trapped.
0b01	This control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL1 to be trapped.
0b10	This control causes execution of these instructions at EL1 and EL0 to be trapped.
0b11	This control does not cause execution of any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:0]

Reserved, RES0.

Accessing CPACR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name CPACR_EL1 or CPACR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to CPACR_EL1 are masked by CPACRMASK_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CPACR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.CPACR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x100];
    else
        X[t, 64] = CPACR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = CPTR_EL2;
    else
        X[t, 64] = CPACR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CPACR_EL1;
```

```
MSR CPACR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.CPACR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x100] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            CPACR_EL1 = (X[t, 64] AND NOT EffectiveCPACRMASK_EL1()) OR (CPACR_EL1 AND
EffectiveCPACRMASK_EL1());
        else
            CPACR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            CPTR_EL2 = (X[t, 64] AND NOT EffectiveCPTRMASK_EL2()) OR (CPTR_EL2 AND
EffectiveCPTRMASK_EL2());
        else
            CPTR_EL2 = X[t, 64];
    else
        CPACR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CPACR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CPACR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x100];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = CPACR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CPACR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR CPACR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x100] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CPACR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CPACR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_SRMASK is implemented

```
MRS <Xt>, CPACRALIAS_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nCPACRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x100];
    else
        X[t, 64] = CPACR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = CPTR_EL2;
    else
        X[t, 64] = CPACR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CPACR_EL1;
```

When FEAT_SRMASK is implemented

```
MSR CPACRALIAS_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nCPACRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x100] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            CPACR_EL1 = (X[t, 64] AND NOT EffectiveCPACRMASK_EL1()) OR (CPACR_EL1 AND
EffectiveCPACRMASK_EL1());
        else
            CPACR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            CPTR_EL2 = (X[t, 64] AND NOT EffectiveCPTRMASK_EL2()) OR (CPTR_EL2 AND
EffectiveCPTRMASK_EL2());
        else
            CPTR_EL2 = X[t, 64];
    else
        CPACR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CPACR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPACRMASK_EL1, Architectural Feature Access Control Masking Register

The CPACRMASK_EL1 characteristics are:

Purpose

Mask register to prevent updates of fields in CPACR_EL1 on writes to CPACR_EL1 or CPACRALIAS_EL1.

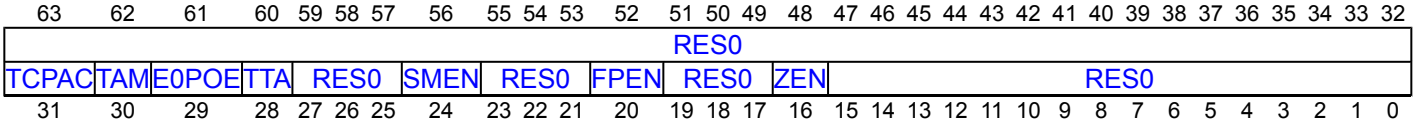
Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CPACRMASK_EL1 are UNDEFINED.

Attributes

CPACRMASK_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

When FEAT_NV2p1 is implemented:

Mask bit for TCPAC.

TCPAC	Meaning
0b0	CPACR_EL1.TCPAC is writeable.
0b1	CPACR_EL1.TCPAC is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TAM, bit [30]

When FEAT_AMUv1 is implemented and FEAT_NV2p1 is implemented:

Mask bit for TAM.

TAM	Meaning
0b0	CPACR_EL1.TAM is writeable.
0b1	CPACR_EL1.TAM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [29]

When FEAT_S1POE is implemented:

Mask bit for E0POE.

E0POE	Meaning
0b0	CPACR_EL1.E0POE is writeable.
0b1	CPACR_EL1.E0POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTA, bit [28]

When System register access to the trace unit registers is implemented:

Mask bit for TTA.

TTA	Meaning
0b0	CPACR_EL1.TTA is writeable.
0b1	CPACR_EL1.TTA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [27:25]

Reserved, RES0.

SMEN, bit [24]

When FEAT_SME is implemented:

Mask bit for SMEN.

SMEN	Meaning
0b0	CPACR_EL1.SMEN is writeable.
0b1	CPACR_EL1.SMEN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [23:21]

Reserved, RES0.

FPEN, bit [20]

Mask bit for FPEN.

FPEN	Meaning
0b0	CPACR_EL1.FPEN is writeable.
0b1	CPACR_EL1.FPEN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [19:17]

Reserved, RES0.

ZEN, bit [16]

When FEAT_SVE is implemented:

Mask bit for ZEN.

ZEN	Meaning
0b0	CPACR_EL1.ZEN is writeable.
0b1	CPACR_EL1.ZEN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:0]

Reserved, RES0.

Accessing CPACRMASK_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name CPACRMASK_EL1 or CPACRMASK_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CPACRMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nCPACRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x320];
    else
        X[t, 64] = CPACRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = CPTRMASK_EL2;
    else
        X[t, 64] = CPACRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CPACRMASK_EL1;
```

```
MSR CPACRMASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nCPACRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x320] = X[t, 64];
    elsif !IsZero(EffectiveCPACRMASK_EL1()) then
        UNDEFINED;
    else
        CPACRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveCPTRMASK_EL2()) then
            UNDEFINED;
        else
            CPTRMASK_EL2 = X[t, 64];
    else
        CPACRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CPACRMASK_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CPACRMASK_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x320];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = CPACRMASK_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CPACRMASK_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR CPACRMASK_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x320] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CPACRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CPACRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPP RCTX, Cache Prefetch Prediction Restriction by Context

The CPP RCTX characteristics are:

Purpose

Cache Prefetch Prediction Restriction by Context applies to all Cache Allocation Resources that predict cache allocations based on information gathered within the target execution context or contexts.

The actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control cache prefetch predictions occurring after the instruction is complete and synchronized.

This instruction applies to all:

- Instruction caches.
- Data caches.
- TLB prefetching hardware used by the executing PE that applies to the supplied context or contexts.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of Cache Allocation Resources so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

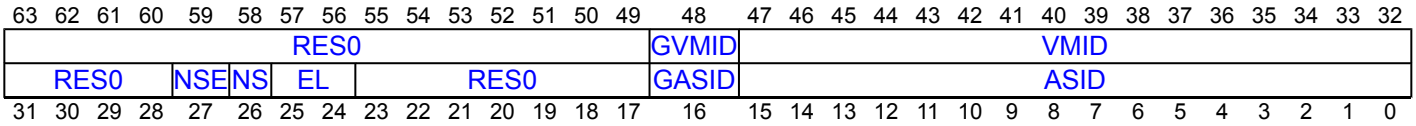
Configuration

This instruction is present only when FEAT_SPECRES is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CPP RCTX are UNDEFINED.

Attributes

CPP RCTX is a 64-bit System instruction.

Field descriptions



Bits [63:49]

Reserved, RES0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 target execution context.
0b1	Applies to all VMIDs for an EL0 or EL1 target execution context.

For target execution contexts other than EL0 and EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and the target execution context is either:

- EL1.
- EL0 when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, this field is treated as the current VMID.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

If the implementation supports 16 bits of VMID, then the upper 8 bits of the VMID must be written to 0 by software when the context being affected only uses 8 bits.

Bits [31:28]

Reserved, RES0.

NSE, bit [27]

When FEAT_RME is implemented:

Together with the NS field, selects the Security state.

For a description of the values derived by evaluating NS and NSE together, see CPP_RCTX.NS.

Otherwise:

Reserved, RES0.

NS, bit [26]

When FEAT_RME is implemented:

Together with the NSE field, selects the Security state.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

Some Effective values are determined by the current Security state:

- When executed in Secure state, the Effective value of NSE is 0.
- When executed in Non-secure state, the Effective value of {NSE, NS} is {0, 1}.
- When executed in Realm state, the Effective value of {NSE, NS} is {1, 1}.

This instruction is treated as a NOP when executed at EL3 and either:

- CPP_RCTX.{NSE, NS} selects a reserved value.
- CPP_RCTX.{NSE, NS} == {1, 0} and CPP_RCTX.EL has a value other than 0b11.

Otherwise:

Security State.

NS	Meaning
0b0	Secure state.
0b1	Non-secure state.

When executed in Non-secure state, the Effective value of NS is 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

If the instruction is executed at an Exception level lower than the specified level, or is specified to apply to a combination of Exception level and Security state that is not implemented, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

GASID	Meaning
0b0	Applies to specified ASID for an EL0 target execution context.
0b1	Applies to all ASIDs for an EL0 target execution context.

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [15:0]

Only applies for an EL0 target execution context and when bit[16] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being affected only uses 8 bits.

Executing CPP RCTX

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
CPP RCTX, <Xt>
```

op0	op1	CRn	CRm	op2

0b01	0b011	0b0111	0b0011	0b111

```
if !(IsFeatureImplemented(FEAT_SPECRES) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.CPPRCTX == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnRCTX == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.RestrictPrediction(X[t, 64], RestrictType_CachePrefetch);
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.CPPRCTX == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.RestrictPrediction(X[t, 64], RestrictType_CachePrefetch);
elsif PSTATE.EL == EL2 then
    AArch64.RestrictPrediction(X[t, 64], RestrictType_CachePrefetch);
elsif PSTATE.EL == EL3 then
    AArch64.RestrictPrediction(X[t, 64], RestrictType_CachePrefetch);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPTR_EL2, Architectural Feature Trap Register (EL2)

The CPTR_EL2 characteristics are:

Purpose

Controls trapping to EL2 of accesses to CPACR, CPACR_EL1, trace, Activity Monitor, SME, Streaming SVE, SVE, and Advanced SIMD and floating-point functionality.

Configuration

AArch64 System register CPTR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HCPTR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CPTR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

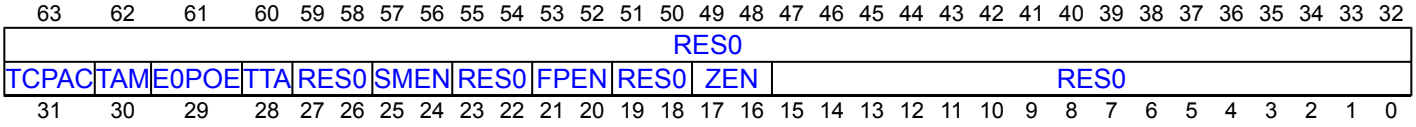
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

CPTR_EL2 is a 64-bit register.

Field descriptions

When ELIsInHost(EL2):



Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

In AArch64 state, traps accesses to CPACR_EL1 from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using EC syndrome value `0x18`.

In AArch32 state, traps accesses to CPACR from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using EC syndrome value `0x03`.

TCPAC	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL1 accesses to CPACR_EL1 and CPACR are trapped to EL2, when EL2 is enabled in the current Security state.

When the Effective value of HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

Note

CPACR_EL1 and CPACR are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Trap Activity Monitor access. Traps EL1 and EL0 accesses to all Activity Monitor registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x18`:
  - AMUSERENR_EL0, AMCFGR_EL0, AMCGCR_EL0, AMCNTENCLR0_EL0, AMCNTENCLR1_EL0, AMCNTENSET0_EL0, AMCNTENSET1_EL0, AMCR_EL0, AMEVCNTR0<n>_EL0, AMEVCNTR1<n>_EL0, AMEVTYPER0<n>_EL0, and AMEVTYPER1<n>_EL0.
- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2 and reported using EC syndrome value `0x03`:
  - AMUSERENR, AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTENSET0, AMCNTENSET1, AMCR, AMEVTYPER0<n>, and AMEVTYPER1<n>.
- In AArch32 state, MRRC or MCRR accesses to AMEVCNTR0<n> and AMEVCNTR1<n>, are trapped to EL2, reported using EC syndrome value `0x04`.

TAM	Meaning
`0b0`	Accesses from EL1 and EL0 to Activity Monitor registers are not trapped.
`0b1`	Accesses from EL1 and EL0 to Activity Monitor registers are trapped to EL2, when EL2 is enabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [29]

When FEAT_S1POE is implemented:

Enable access to POR_EL0.

Traps EL0 accesses to POR_EL0 to EL2, from AArch64 state only. The exception is reported using EC syndrome value `0x18`.

E0POE	Meaning
`0b0`	This control causes EL0 access to POR_EL0 to be trapped.
`0b1`	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTA, bit [28]

When System register access to the trace unit registers is implemented:

Traps System register accesses to all implemented trace registers from both Execution states to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to trace registers with op0=2, op1=1, and CRn<`0b1000` are trapped to EL2, reported using EC syndrome value `0x18`.

- In AArch32 state, MRC or MCR accesses to trace registers with cpnum=14, opc1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x05.

TTA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt at EL0, EL1 or EL2, to execute a System register access to an implemented trace register is trapped to EL2, when EL2 is enabled in the current Security state, unless HCR_EL2.TGE is 0 and it is trapped by CPACR.NSTRCDIS or CPACR_EL1.TTA. When HCR_EL2.TGE is 1, any attempt at EL0 or EL2 to execute a System register access to an implemented trace register is trapped to EL2, when EL2 is enabled in the current Security state.

Note

The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the trace unit implements FEAT_ETMv4 or ETE, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.TTA is 1.

EL2 does not provide traps on trace register accesses through the optional Memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [27:26]

Reserved, RES0.

SMEN, bits [25:24]

When FEAT_SME is implemented:

Traps execution at EL2, EL1, and EL0 of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the SVCR, SMCR_EL1, or SMCR_EL2 System registers to EL2, when EL2 is enabled in the current Security state.

When instructions that directly access the SVCR System register are trapped with reference to this control, the MSR SVCRSM, MSR SVCRZA, and MSR SVCRSMZA instructions are also trapped.

The exception is reported using EC syndrome value 0x1D, with an ISS code of 0x0000000.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPTR_EL2.SMEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

SMEN	Meaning
0b00	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b01	When HCR_EL2.TGE is 0, this control does not cause execution of any instructions to be trapped. When HCR_EL2.TGE is 1, this control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL2 to be trapped.
0b10	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b11	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [23:22]

Reserved, RES0.

FPEN, bits [21:20]

Traps execution at EL2, EL1, and EL0 of instructions that access the Advanced SIMD and floating-point registers from both Execution states to EL2, when EL2 is enabled in the current Security state. The exception is reported using EC syndrome value 0x07.

Traps execution at EL2, EL1, and EL0 of SME and SVE instructions to EL2, when EL2 is enabled in the current Security state. The exception is reported using EC syndrome value 0x07.

When FEAT_FPMR is implemented, traps execution at EL2, EL1, and EL0 of instructions that access FPMR to EL2, when EL2 is enabled in the current Security state. The exception is reported using EC syndrome value 0x07

A trap taken as a result of CPTR_EL2.SMEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

A trap taken as a result of CPTR_EL2.ZEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

FPEN	Meaning
0b00	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b01	When HCR_EL2.TGE is 0, this control does not cause execution of any instructions to be trapped. When HCR_EL2.TGE is 1, this control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL2 to be trapped.
0b10	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b11	This control does not cause execution of any instructions to be trapped.

Writes to MVFR0, MVFR1, and MVFR2 from EL1 or higher are CONSTRAINED UNPREDICTABLE and whether these accesses can be trapped by this control depends on implemented CONSTRAINED UNPREDICTABLE behavior.

Note

- Attempts to write to the FPSID count as use of the registers for accesses from EL1 or higher.
- Accesses from EL0 to FPSID, MVFR0, MVFR1, MVFR2, and FPEXC are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.FPEN is not 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:18]

Reserved, RES0.

ZEN, bits [17:16]

When FEAT_SVE is implemented:

Traps execution at EL2, EL1, and EL0 of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the ZCR_EL1 or ZCR_EL2 System registers to EL2, when EL2 is enabled in the current Security state.

The exception is reported using EC syndrome value `0x19`.

A trap taken as a result of CPTR_EL2.ZEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

ZEN	Meaning
`0b00`	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
`0b01`	When HCR_EL2.TGE is 0, this control does not cause execution of any instructions to be trapped. When HCR_EL2.TGE is 1, this control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL2 to be trapped.
`0b10`	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
`0b11`	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

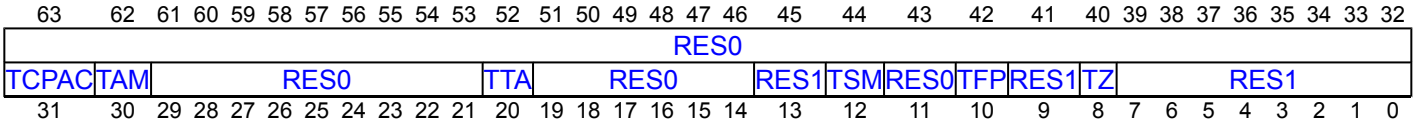
Otherwise:

Reserved, RES0.

Bits [15:0]

Reserved, RES0.

Otherwise:



This format applies in all Armv8.0 implementations.

Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

In AArch64 state, traps accesses to CPACR_EL1 from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using EC syndrome value `0x18`.

In AArch32 state, traps accesses to CPACR from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using EC syndrome value `0x03`.

TCPAC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the following registers are trapped to EL2, when EL2 is enabled in the current Security state: • CPACR_EL1. • CPACR.

When HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

Note

CPACR_EL1 and CPACR are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Trap Activity Monitor access. Traps EL1 and EL0 accesses to all Activity Monitor registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
  - AMUSERENR_EL0, AMCFGR_EL0, AMCGCR_EL0, AMCNTENCLR0_EL0, AMCNTENCLR1_EL0, AMCNTENSET0_EL0, AMCNTENSET1_EL0, AMCR_EL0, AMEVCNTR0<n>_EL0, AMEVCNTR1<n>_EL0, AMEVTYPER0<n>_EL0, and AMEVTYPER1<n>_EL0.
- In AArch32 state, MCR or MRC accesses to the following registers are trapped to EL2 and reported using EC syndrome value 0x03:
  - AMUSERENR, AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTENSET0, AMCNTENSET1, AMCR, AMEVTYPER0<n>, and AMEVTYPER1<n>.
- In AArch32 state, MCRR or MRRC accesses to AMEVCNTR0<n> and AMEVCNTR1<n>, are trapped to EL2, reported using EC syndrome value 0x04.

TAM	Meaning
0b0	Accesses from EL1 and EL0 to Activity Monitor registers are not trapped.
0b1	Accesses from EL1 and EL0 to Activity Monitor registers are trapped to EL2, when EL2 is enabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

When System register access to the trace unit registers is implemented:

Traps System register accesses to all implemented trace registers from both Execution states to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to trace registers with op0=2, op1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x18.

- In AArch32 state, MRC or MCR accesses to trace registers with cpnum=14, opc1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x05.

TTA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt at EL0, EL1, or EL2, to execute a System register access to an implemented trace register is trapped to EL2, when EL2 is enabled in the current Security state, unless it is trapped by one of the following controls: • CPACR_EL1.TTA. • CPACR.TRCDIS.

Note

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the trace unit implements FEAT_ETMv4, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.TTA is 1.
- EL2 does not provide traps on trace register accesses through the optional memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:14]

Reserved, RES0.

Bit [13]

Reserved, RES1.

TSM, bit [12]

When FEAT_SME is implemented:

Traps execution at EL2, EL1, and EL0 of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the SVCR, SMCR_EL1, or SMCR_EL2 System registers to EL2, when EL2 is enabled in the current Security state.

When instructions that directly access the SVCR System register are trapped with reference to this control, the MSR SVCRSM, MSR SVCRZA, and MSR SVCRSMZA instructions are also trapped.

The exception is reported using EC syndrome value 0x1D, with an ISS code of 0x0000000.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPTR_EL2.TSM has precedence over a trap taken as a result of CPTR_EL2.TFP.

TSM	Meaning
0b0	This control does not cause execution of any instructions to be trapped.
0b1	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [11]

Reserved, RES0.

TFP, bit [10]

Traps execution of instructions which access the Advanced SIMD and floating-point functionality, from both Execution states to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x07`:
  - FPCR, FPSR, FPEXC32_EL2, and any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-31 registers.
  - If FEAT_FPMR is implemented, FPMR.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x07`:
  - MVFR0, MVFR1, MVFR2, FPSCR, FPEXC, and any of the SIMD and floating-point registers Q0-15, including their views as D0-D31 registers or S0-31 registers. For the purposes of this trap, the architecture defines a VMSR access to FPSID from EL1 or higher as an access to a SIMD and floating-point register. Otherwise, permitted VMSR accesses to FPSID are ignored.

Traps execution at the same Exception levels of SME and SVE instructions to EL2, when EL2 is enabled in the current Security state. The exception is reported using EC syndrome value `0x07`.

A trap taken as a result of CPTR_EL2.TSM has precedence over a trap taken as a result of CPTR_EL2.TFP.

A trap taken as a result of CPTR_EL2.TZ has precedence over a trap taken as a result of CPTR_EL2.TFP.

TFP	Meaning
`0b0`	This control does not cause execution of any instructions to be trapped.
`0b1`	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.

Note

FPEXC32_EL2 is not accessible from EL0 using AArch64.

FPSID, MVFR0, MVFR1, and FPEXC are not accessible from EL0 using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES1.

TZ, bit [8]

When FEAT_SVE is implemented:

Traps execution at EL2, EL1, and EL0 of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the ZCR_EL2 or ZCR_EL1 System registers to EL2, when EL2 is enabled in the current Security state.

The exception is reported using EC syndrome value `0x19`.

A trap taken as a result of CPTR_EL2.TZ has precedence over a trap taken as a result of CPTR_EL2.TFP.

TZ	Meaning
`0b0`	This control does not cause execution of any instructions to be trapped.
`0b1`	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bits [7:0]

Reserved, RES1.

Accessing CPTR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name CPTR_EL2 or CPACR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to CPTR_EL2 are masked by CPTRMASK_EL2.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CPTR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = CPTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CPTR_EL2;
```

```
MSR CPTR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            CPTR_EL2 = (X[t, 64] AND NOT EffectiveCPTRMASK_EL2()) OR (CPTR_EL2 AND
EffectiveCPTRMASK_EL2());
        else
            CPTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CPTR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, CPACR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.CPACR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x100];
    else
        X[t, 64] = CPACR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = CPTR_EL2;
    else
        X[t, 64] = CPACR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CPACR_EL1;
```

When FEAT_VHE is implemented

```
MSR CPACR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.CPACR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x100] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            CPACR_EL1 = (X[t, 64] AND NOT EffectiveCPACRMASK_EL1()) OR (CPACR_EL1 AND
EffectiveCPACRMASK_EL1());
        else
            CPACR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            CPTR_EL2 = (X[t, 64] AND NOT EffectiveCPTRMASK_EL2()) OR (CPTR_EL2 AND
EffectiveCPTRMASK_EL2());
        else
            CPTR_EL2 = X[t, 64];
    else
        CPACR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CPACR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPTR_EL3, Architectural Feature Trap Register (EL3)

The CPTR_EL3 characteristics are:

Purpose

Controls trapping to EL3 of accesses to CPACR, CPACR_EL1, HCPTR, CPTR_EL2, trace, Activity Monitor, SME, Streaming SVE, SVE, and Advanced SIMD and floating-point functionality.

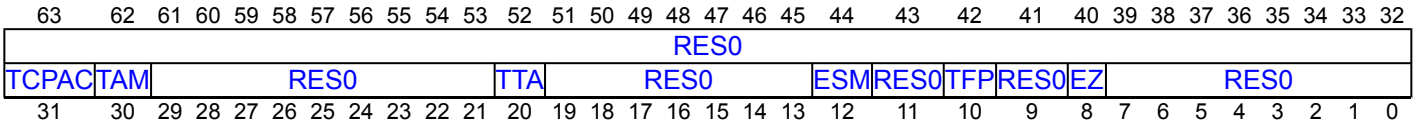
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CPTR_EL3 are UNDEFINED.

Attributes

CPTR_EL3 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

Traps all of the following to EL3, from both Execution states and any Security state.

- EL2 accesses to CPTR_EL2, reported using EC syndrome value `0x18`, or HCPTR, reported using EC syndrome value `0x03`.
- EL2 and EL1 accesses to CPACR_EL1 reported using EC syndrome value `0x18`, or CPACR reported using EC syndrome value `0x03`.

When CPTR_EL3.TCPAC is:

TCPAC	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL2 accesses to the CPTR_EL2 or HCPTR, and EL2 and EL1 accesses to the CPACR_EL1 or CPACR, are trapped to EL3, unless they are trapped by CPTR_EL2.TCPAC.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Trap Activity Monitor access. Traps EL2, EL1, and EL0 accesses to all Activity Monitor registers to EL3.

Accesses to the Activity Monitors registers are trapped as follows:

- In AArch64 state, the following registers are trapped to EL3 and reported using EC syndrome value `0x18`:

        - AMUSERENR_EL0, AMCFGR_EL0, AMCGCR_EL0, AMCNTENCLR0_EL0, AMCNTENCLR1_EL0, AMCNTENSET0_EL0, AMCNTENSET1_EL0, AMCR_EL0, AMEVCNTR0<n>_EL0, AMEVCNTR1<n>_EL0, AMEVTYPER0<n>_EL0, and AMEVTYPER1<n>_EL0.

    - In AArch32 state, accesses with MRC or MCR to the following registers reported using EC syndrome value `0x03`:

        - AMUSERENR, AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTENSET0, AMCNTENSET1, AMCR, AMEVTYPER0<n>, and AMEVTYPER1<n>.

    - In AArch32 state, accesses with MRRC or MCRR to the following registers, reported using EC syndrome value `0x04`:

        - AMEVCNTR0<n>, AMEVCNTR1<n>.

TAM	Meaning
`0b0`	Accesses from EL2, EL1, and EL0 to Activity Monitor registers are not trapped.
`0b1`	Accesses from EL2, EL1, and EL0 to Activity Monitor registers are trapped to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

When System register access to the trace unit registers is implemented:

Traps System register accesses. Accesses to the trace registers, from all Exception levels, any Security state, and both Execution states are trapped to EL3 as follows:

- In AArch64 state, Trace registers with op0=2, op1=1, and CRn<`0b1000` are trapped to EL3 and reported using EC syndrome value `0x18`.
- In AArch32 state, accesses using MCR or MRC to the Trace registers with cpnum=14, opc1=1, and CRn<`0b1000` are reported using EC syndrome value `0x05`.

TTA	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Any System register access to the trace registers is trapped to EL3, unless it is trapped by CPACR.TRCDIS, CPACR_EL1.TTA, or CPTR_EL2.TTA.

Note

The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than this trap exception.

EL3 does not provide traps on trace register accesses through the Memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, no side-effects occur before the exception is taken, see 'Configurable instruction controls'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:13]

Reserved, RES0.

ESM, bit [12]

When FEAT_SME is implemented:

Traps execution of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the SMCR_EL1, SMCR_EL2, SMCR_EL3, SMPRI_EL1, SMPRIMAP_EL2, or SVCR System registers, from all Exception levels and any Security state, to EL3.

When instructions that directly access the SVCR System register are trapped with reference to this control, the `MSR SVCRSM`, `MSR SVCRZA`, and `MSR SVCRSMZA` instructions are also trapped.

When direct accesses to SMPRI_EL1 and SMPRIMAP_EL2 are trapped, the exception is reported using EC syndrome value `0x18`. Otherwise, the exception is reported using EC syndrome value `0x1D`, with an ISS code of `0x0000000`.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPTR_EL3.ESM has precedence over a trap taken as a result of CPTR_EL3.TFP.

ESM	Meaning
`0b0`	This control causes execution of these instructions at all Exception levels to be trapped.
`0b1`	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES0.

TFP, bit [10]

Traps execution of instructions which access the Advanced SIMD and floating-point functionality, from all Exception levels, any Security state, and both Execution states, to EL3.

This includes the following registers, all reported using EC syndrome value `0x07`:

- FPCR, FPSR, FPEXC32_EL2, and any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-S31 registers.
- If FEAT_FPMR is implemented, FPMR.
- MVFR0, MVFR1, MVFR2, FPSCR, FPEXC, and any of the SIMD and floating-point registers Q0-Q15, including their views as D0-D31 registers or S0-S31 registers.
- VMSR accesses to FPSID.

Permitted VMSR accesses to FPSID are ignored, but for the purposes of this trap the architecture defines a VMSR access to the FPSID from EL1 or higher as an access to a SIMD and floating-point register.

Traps execution at all Exception levels of SME and SVE instructions to EL3 from any Security state. The exception is reported using EC syndrome value `0x07`.

A trap taken as a result of CPTR_EL3.ESM has precedence over a trap taken as a result of CPTR_EL3.TFP.

A trap taken as a result of CPTR_EL3.EZ has precedence over a trap taken as a result of CPTR_EL3.TFP.

TFP	Meaning
0b0	This control does not cause execution of any instructions to be trapped.
0b1	This control causes execution of these instructions at all Exception levels to be trapped.

Note

FPEXC32_EL2 is not accessible from EL0 using AArch64.

FPSID, MVFR0, MVFR1, and FPEXC are not accessible from EL0 using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

EZ, bit [8]
When FEAT_SVE is implemented:

Traps execution of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the ZCR_EL3, ZCR_EL2, or ZCR_EL1 System registers, from all Exception levels and any Security state, to EL3.

The exception is reported using EC syndrome value 0x19.

A trap taken as a result of CPTR_EL3.EZ has precedence over a trap taken as a result of CPTR_EL3.TFP.

EZ	Meaning
0b0	This control causes execution of these instructions at all Exception levels to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [7:0]

Reserved, RES0.

Accessing CPTR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CPTR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CPTR_EL3;
```

```
MSR CPTR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    CPTR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPTRMASK_EL2, Architectural Feature Trap Masking Register

The CPTRMASK_EL2 characteristics are:

Purpose

Mask register to prevent updates of fields in CPTR_EL2 on writes.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CPTRMASK_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

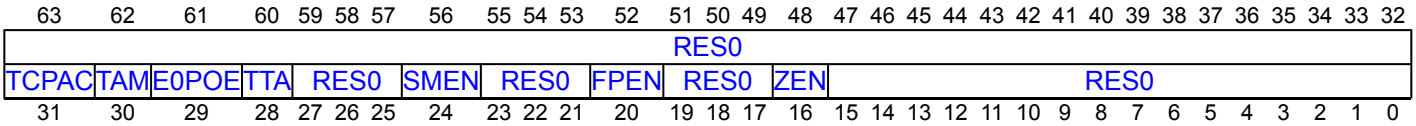
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

CPTRMASK_EL2 is a 64-bit register.

Field descriptions

When ELIsInHost(EL2):



Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

Mask bit for TCPAC.

TCPAC	Meaning
0b0	CPTR_EL2.TCPAC is writeable.
0b1	CPTR_EL2.TCPAC is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Mask bit for TAM.

TAM	Meaning
0b0	CPTR_EL2.TAM is writeable.
0b1	CPTR_EL2.TAM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [29]

When FEAT_S1POE is implemented:

Mask bit for E0POE.

E0POE	Meaning
0b0	CPTR_EL2.E0POE is writeable.
0b1	CPTR_EL2.E0POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTA, bit [28]

When System register access to the trace unit registers is implemented:

Mask bit for TTA.

TTA	Meaning
0b0	CPTR_EL2.TTA is writeable.
0b1	CPTR_EL2.TTA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [27:25]

Reserved, RES0.

SMEN, bit [24]

When FEAT_SME is implemented:

Mask bit for SMEN.

SMEN	Meaning
0b0	CPTR_EL2.SMEN is writeable.
0b1	CPTR_EL2.SMEN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [23:21]

Reserved, RES0.

FPEN, bit [20]

Mask bit for FPEN.

FPEN	Meaning
0b0	CPTR_EL2.FPEN is writeable.
0b1	CPTR_EL2.FPEN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [19:17]

Reserved, RES0.

ZEN, bit [16]

When FEAT_SVE is implemented:

Mask bit for ZEN.

ZEN	Meaning
0b0	CPTR_EL2.ZEN is writeable.
0b1	CPTR_EL2.ZEN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

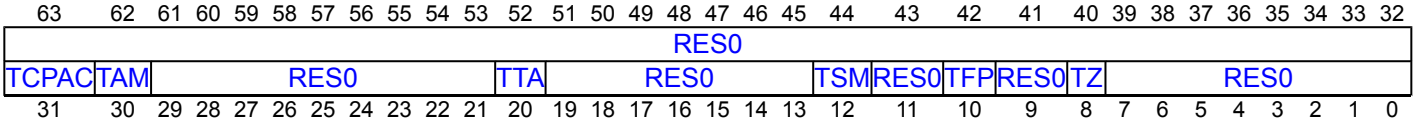
Otherwise:

Reserved, RES0.

Bits [15:0]

Reserved, RES0.

Otherwise:



This format applies in all Armv8.0 implementations.

Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

Mask bit for TCPAC.

TCPAC	Meaning
0b0	CPTR_EL2.TCPAC is writeable.
0b1	CPTR_EL2.TCPAC is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Mask bit for TAM.

TAM	Meaning
0b0	CPTR_EL2.TAM is writeable.
0b1	CPTR_EL2.TAM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

Mask bit for TTA.

TTA	Meaning
0b0	CPTR_EL2.TTA is writeable.
0b1	CPTR_EL2.TTA is not writeable.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [19:13]

Reserved, RES0.

TSM, bit [12]

When FEAT_SME is implemented:

Mask bit for TSM.

TSM	Meaning
0b0	CPTR_EL2.TSM is writeable.
0b1	CPTR_EL2.TSM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES0.

TFP, bit [10]

Mask bit for TFP.

TFP	Meaning
0b0	CPTR_EL2.TFP is writeable.
0b1	CPTR_EL2.TFP is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

TZ, bit [8]

When FEAT_SVE is implemented:

Mask bit for TZ.

TZ	Meaning
0b0	CPTR_EL2.TZ is writeable.
0b1	CPTR_EL2.TZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [7:0]

Reserved, RES0.

Accessing CPTRMASK_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name CPTRMASK_EL2 or CPTRMASK_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CPTRMASK_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = CPTRMASK_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CPTRMASK_EL2;
```

```
MSR CPTRMASK_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsZero(EffectiveCPTRMASK_EL2()) then
        UNDEFINED;
    else
        CPTRMASK_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CPTRMASK_EL2 = X[t, 64];
```

MRS <Xt>, CPACRMASK_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nCPACRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x320];
    else
        X[t, 64] = CPACRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = CPTRMASK_EL2;
    else
        X[t, 64] = CPACRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CPACRMASK_EL1;
```

```
MSR CPACRMASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nCPACRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x320] = X[t, 64];
    elsif !IsZero(EffectiveCPACRMASK_EL1()) then
        UNDEFINED;
    else
        CPACRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveCPTRMASK_EL2()) then
            UNDEFINED;
        else
            CPTRMASK_EL2 = X[t, 64];
    else
        CPACRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CPACRMASK_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CSSELR_EL1, Cache Size Selection Register

The CSSELR_EL1 characteristics are:

Purpose

Selects the current Cache Size ID Register, CCSIDR_EL1, by specifying the required cache level and the cache type (either instruction or data cache).

Configuration

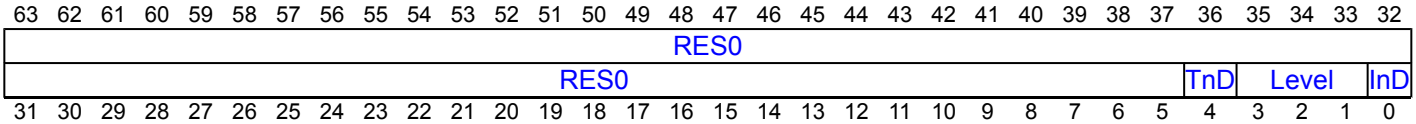
AArch64 System register CSSELR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CSSELR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CSSELR_EL1 are UNDEFINED.

Attributes

CSSELR_EL1 is a 64-bit register.

Field descriptions



Bits [63:5]

Reserved, RES0.

TnD, bit [4]

When FEAT_MTE2 is implemented:

Allocation Tag not Data bit.

TnD	Meaning
0b0	Data, Instruction or Unified cache.
0b1	Separate Allocation Tag cache.

When CSSELR_EL1.InD == 1, this bit is RES0.

If CSSELR_EL1.{TnD, Level, InD} is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR_EL1 is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Level, bits [3:1]

Cache level of required cache.

Level	Meaning
0b000	Level 1 cache.
0b001	Level 2 cache.
0b010	Level 3 cache.
0b011	Level 4 cache.
0b100	Level 5 cache.
0b101	Level 6 cache.
0b110	Level 7 cache.

All other values are reserved.

If CSSELR_EL1.{TnD, Level, InD} is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR_EL1 is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

InD, bit [0]

Instruction not Data bit.

InD	Meaning
0b0	Data or unified cache.
0b1	Instruction cache.

If CSSELR_EL1.{TnD, Level, InD} is programmed to a cache level that is not implemented, then a read of CSSELR_EL1 is CONSTRAINED UNPREDICTABLE, and returns UNKNOWN values for CSSELR_EL1.{Level, InD}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CSSELR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CSSELR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b010	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_EVT) && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.CSSELR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = CSSELR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CSSELR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CSSELR_EL1;
```

```
MSR CSSELR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b010	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_EVT) && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.CSSELR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        CSSELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    CSSELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CSSELR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTR_EL0, Cache Type Register

The CTR_EL0 characteristics are:

Purpose

Provides information about the architecture of the caches.

Configuration

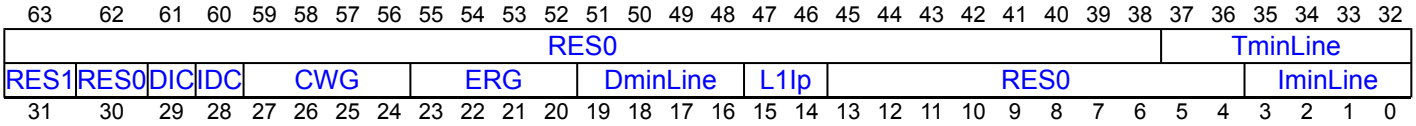
AArch64 System register CTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register CTR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CTR_EL0 are UNDEFINED.

Attributes

CTR_EL0 is a 64-bit register.

Field descriptions



Bits [63:38]

Reserved, RES0.

TminLine, bits [37:32]

When FEAT_MTE2 is implemented:

Tag minimum Line. Log_2 of the number of words covered by Allocation Tags in the smallest cache line of all caches which can contain Allocation tags that are controlled by the PE.

Note

- For an implementation with cache lines containing 64 bytes of data and 4 Allocation Tags, this will be $log_2(64/4) = 4$.
- For an implementation with Allocations Tags in separate cache lines of 128 Allocation Tags per line, this will be $log_2(128*16/4) = 9$.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES1.

Bit [30]

Reserved, RES0.

DIC, bit [29]

Instruction cache invalidation requirements for data to instruction coherence.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DIC	Meaning
0b0	Instruction cache invalidation to the Point of Unification is required for data to instruction coherence.
0b1	Instruction cache invalidation to the Point of Unification is not required for data to instruction coherence.

All PEs in the same Inner Shareable shareability domain must have a common value of this field.

Note

If CTR_EL0.DIC is 1, instruction cache maintenance is not required after overwriting instructions, including for different VA aliases of the affected Locations, regardless of the value of CTR_EL0.L1Ip.

Access to this field is **RO**.

IDC, bit [28]

Data cache clean requirements for instruction to data coherence. The meaning of this bit is:

The value of this field is an IMPLEMENTATION DEFINED choice of:

IDC	Meaning
0b0	Data cache clean to the Point of Unification is required for instruction to data coherence, unless CLIDR_EL1.LoC == 0b000 or (CLIDR_EL1.LoUIS == 0b000 and CLIDR_EL1.LoUU == 0b000).
0b1	Data cache clean to the Point of Unification is not required for instruction to data coherence.

If CTR_EL0.DIC is 1 then the value reported in this field must also be 1.

The Effective value of IDC is 1 if any of the following are true:

- CTR_EL0.IDC == 1.
- CLIDR_EL1.LoC == 0b000.
- CLIDR_EL1.LoUIS == 0b000 and CLIDR_EL1.LoUU == 0b000.

All PEs in the same Inner Shareable shareability domain must have a common Effective value of IDC.

Access to this field is **RO**.

CWG, bits [27:24]

Cache writeback granule. Log_2 of the number of words of the maximum size of memory that can be overwritten as a result of the eviction of a cache entry that has had a memory location in it modified.

A value of 0b0000 indicates that this register does not provide Cache writeback granule information and either:

- The architectural maximum of 512 words (2KB) must be assumed.
- The Cache writeback granule can be determined from maximum cache line size encoded in the Cache Size ID Registers.

Values greater than 0b1001 are reserved.

Arm recommends that an implementation that does not support cache write-back implements this field as 0b0001. This applies, for example, to an implementation that supports only write-through caches.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ERG, bits [23:20]

Exclusives reservation granule, and, if FEAT_TME is implemented, transactional reservation granule. Log_2 of the number of words of the maximum size of the reservation granule for the Load-Exclusive and Store-Exclusive instructions, and, if FEAT_TME is implemented, for detecting transactional conflicts.

A value of `0b0000` indicates that this register does not provide granule information and the architectural maximum of 512 words (2KB) must be assumed.

Value `0b0001` and values greater than `0b1001` are reserved.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

DminLine, bits [19:16]

Log_2 of the number of words in the smallest cache line of all the data caches and unified caches that are controlled by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

L1Ip, bits [15:14]

Level 1 instruction cache policy. Indicates the indexing and tagging policy for the L1 instruction cache.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1Ip	Meaning
`0b00`	Reserved.
`0b01`	ASID-tagged Virtual Index, Virtual Tag (AIVIVT).
`0b10`	Virtual Index, Physical Tag (VIPT).
`0b11`	Physical Index, Physical Tag (PIPT).

From Armv8.0, the value `0b01` is reserved.

Access to this field is **RO**.

Bits [13:4]

Reserved, RES0.

IminLine, bits [3:0]

Log_2 of the number of words in the smallest cache line of all the instruction caches that are controlled by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CTR_EL0
```

op0	op1	CRn	CRm	op2

0b11	0b011	0b0000	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCT == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGRTR_EL2.CTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = CTR_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.CTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = CTR_EL0;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CTR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CTR_EL0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CurrentEL, Current Exception Level

The CurrentEL characteristics are:

Purpose

Holds the current Exception level.

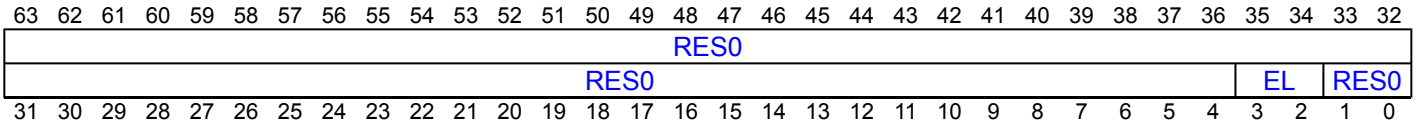
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CurrentEL are UNDEFINED.

Attributes

CurrentEL is a 64-bit register.

Field descriptions



Bits [63:4]

Reserved, RES0.

EL, bits [3:2]

Current Exception level.

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

When the Effective value of HCR_EL2.NV is 1, EL1 read accesses to the CurrentEL register return the value of 0b10 in this field.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '01'.
  - When the highest implemented Exception level is EL2, this field resets to '10'.
  - Otherwise, this field resets to '11'.

Bits [1:0]

Reserved, RES0.

Accessing CurrentEL

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CurrentEL
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        X[t, 64] = Zeros(60):'10':Zeros(2);
    else
        X[t, 64] = Zeros(60):PSTATE.EL:Zeros(2);
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(60):PSTATE.EL:Zeros(2);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(60):PSTATE.EL:Zeros(2);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DACR32_EL2, Domain Access Control Register

The DACR32_EL2 characteristics are:

Purpose

Allows access to the AArch32 DACR register from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register DACR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register DACR[31:0].

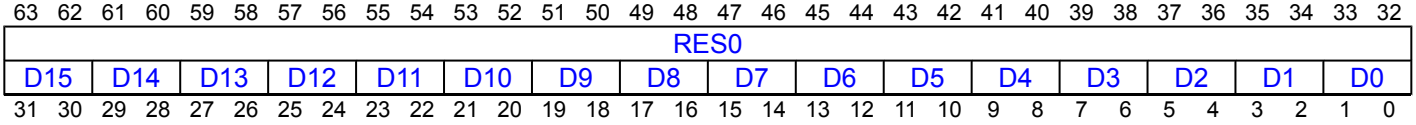
This register is present only when FEAT_AA32EL1 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DACR32_EL2 are UNDEFINED.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

Attributes

DACR32_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

D<n>, bits [2n+1:2n], for n = 15 to 0

Domain n access permission, where n = 0 to 15. Permitted values are:

D<n>	Meaning
0b00	No access. Any access to the domain generates a Domain fault.
0b01	Client. Accesses are checked against the permission bits in the translation tables.
0b11	Manager. Accesses are not checked against the permission bits in the translation tables.

The value 0b10 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DACR32_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DACR32_EL2
```

op0	op1	CRn	CRm	op2

0b11	0b100	0b0011	0b0000	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = DACR32_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DACR32_EL2;
```

```
MSR DACR32_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0000	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    DACR32_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DACR32_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DAIF, Interrupt Mask Bits

The DAIF characteristics are:

Purpose

Allows access to the interrupt mask bits.

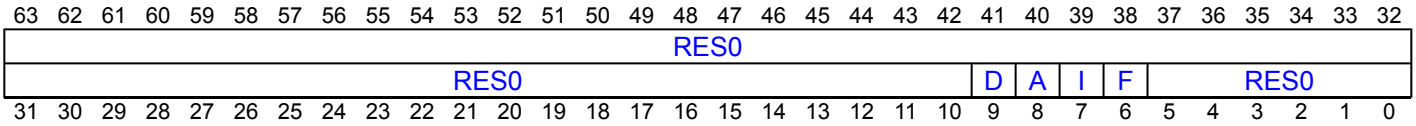
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DAIF are UNDEFINED.

Attributes

DAIF is a 64-bit register.

Field descriptions



Bits [63:10]

Reserved, RES0.

D, bit [9]

Process state D mask.

D	Meaning
0b0	Watchpoint, Breakpoint, and Software Step exceptions targeted at the current Exception level are not masked.
0b1	Watchpoint, Breakpoint, and Software Step exceptions targeted at the current Exception level are masked.

When the target Exception level of the debug exception is higher than the current Exception level, the exception is not masked by this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

A, bit [8]

SError exception mask bit.

A	Meaning
0b0	Exception not masked.
0b1	Exception masked.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

I, bit [7]

IRQ mask bit.

I	Meaning
0b0	Exception not masked.
0b1	Exception masked.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

F, bit [6]

FIQ mask bit.

F	Meaning
0b0	Exception not masked.
0b1	Exception masked.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

Bits [5:0]

Reserved, RES0.

Accessing DAIF

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DAIF

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if ELIsInHost(EL0) || SCTLR_EL1.UMA == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        X[t, 64] = Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
elsif PSTATE.EL == EL1 then
    X[t, 64] = Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
```

MSR DAIF, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if ELIsInHost(EL0) || SCTLR_EL1.UMA == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        PSTATE.<D,A,I,F> = X[t, 64]<9:6>;
elsif PSTATE.EL == EL1 then
    PSTATE.<D,A,I,F> = X[t, 64]<9:6>;
elsif PSTATE.EL == EL2 then
    PSTATE.<D,A,I,F> = X[t, 64]<9:6>;
elsif PSTATE.EL == EL3 then
    PSTATE.<D,A,I,F> = X[t, 64]<9:6>;
```

MSR DAIFSet, #<imm>

op0	op1	CRn	op2
0b00	0b011	0b0100	0b110

MSR DAIFClr, #<imm>

op0	op1	CRn	op2
0b00	0b011	0b0100	0b111

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGAUTHSTATUS_EL1, Debug Authentication Status Register

The DBGAUTHSTATUS_EL1 characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

Configuration

AArch64 System register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGAUTHSTATUS[31:0].

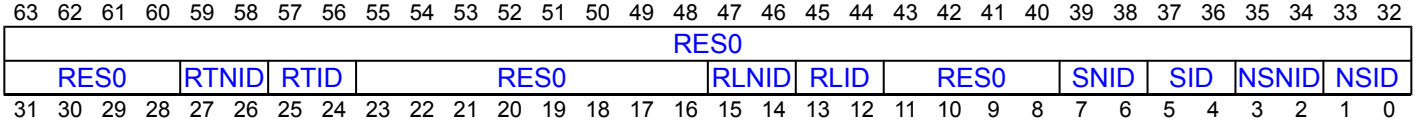
AArch64 System register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to External register DBGAUTHSTATUS_EL1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGAUTHSTATUS_EL1 are UNDEFINED.

Attributes

DBGAUTHSTATUS_EL1 is a 64-bit register.

Field descriptions



Bits [63:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTID.

RTID, bits [25:24]

Root invasive debug.

RTID	Meaning
0b00	Not implemented.
0b10	Implemented and disabled. ExternalRootInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalRootInvasiveDebugEnabled() == TRUE.

All other values are reserved.

If FEAT_RME is not implemented, the only permitted value is 0b00.

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLID.

RLID, bits [13:12]

Realm invasive debug.

RLID	Meaning
0b00	Not implemented.
0b10	Implemented and disabled. ExternalRealmInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalRealmInvasiveDebugEnabled() == TRUE.

All other values are reserved.

If FEAT_RME is not implemented, the only permitted value is 0b00.

Bits [11:8]

Reserved, RES0.

SNID, bits [7:6]

When FEAT_Debugv8p4 is implemented:

Secure non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.SID.

Otherwise:

Secure non-invasive debug.

SNID	Meaning
0b00	Secure state is not implemented.
0b10	Implemented and disabled. ExternalSecureNoninvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalSecureNoninvasiveDebugEnabled() == TRUE.

All other values are reserved.

SID, bits [5:4]

Secure invasive debug.

SID	Meaning
0b00	Secure state is not implemented.
0b10	Implemented and disabled. ExternalSecureInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalSecureInvasiveDebugEnabled() == TRUE.

All other values are reserved.

NSNID, bits [3:2]

When FEAT_Debugv8p4 is implemented:

Non-secure non-invasive debug.

NSNID	Meaning
0b00	Non-secure state is not implemented.
0b11	Implemented and enabled. EL3 is implemented or the Effective value of SCR_EL3.NS is 1.

All other values are reserved.

Otherwise:

Non-secure non-invasive debug.

NSNID	Meaning
0b00	Non-secure state is not implemented.
0b10	Implemented and disabled. ExternalNoninvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE.

All other values are reserved.

NSID, bits [1:0]

Non-secure invasive debug.

NSID	Meaning
0b00	Non-secure state is not implemented.
0b10	Implemented and disabled. ExternalInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalInvasiveDebugEnabled() == TRUE.

All other values are reserved.

Accessing DBGAUTHSTATUS_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGAUTHSTATUS_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0111	0b1110	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.DBGAUTHSTATUS_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGAUTHSTATUS_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGAUTHSTATUS_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DBGAUTHSTATUS_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGBCR<n>_EL1, Debug Breakpoint Control Registers, n = 0 - 63

The DBGBCR<n>_EL1 characteristics are:

Purpose

Holds control information for a breakpoint. Forms breakpoint n together with value register DBGBVR<n>_EL1.

Configuration

AArch64 System register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGBCR<n>[31:0].

AArch64 System register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to External register DBGBCR<n>_EL1[31:0].

AArch64 System register DBGBCR<n>_EL1 bits [63:32] are architecturally mapped to External register DBGBCR<n>_EL1[63:32] when FEAT_Debugv8p9 is implemented.

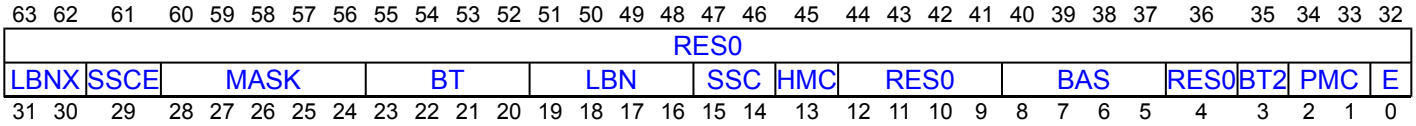
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGBCR<n>_EL1 are UNDEFINED.

If breakpoint n is not implemented, accesses to this register are UNDEFINED.

Attributes

DBGBCR<n>_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

LBNX, bits [31:30]

When FEAT_Debugv8p9 is implemented:

Linked Breakpoint Number.

For Linked address matching breakpoints, with DBGBCR<n>_EL1.LBN, specifies the index of the breakpoint linked to.

For all other breakpoint types, this field is ignored and reads of the register return an UNKNOWN value.

This field extends DBGBCR<n>_EL1.LBN to support up to 64 implemented breakpoints.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSCE, bit [29]

When FEAT_RME is implemented:

Security State Control Extended.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MASK, bits [28:24]

When FEAT_BWE is implemented:

Address Mask. Only address ranges up to 2GB can be watched using a single mask.

MASK	Meaning
0b00000	No mask.
0b00011..0b11111	Number of address bits masked.

All other values are reserved.

Indicates the number of masked address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b11111 masking 31 address bits (0x7FFFFFFF mask for address).

If any of the following apply then the behavior of the breakpoint is CONSTRAINED UNPREDICTABLE:

- DBGBCR<n>_EL1.MASK is a reserved value.
- DBGBCR<n>_EL1.MASK is zero, DBGBCR<n>_EL1.{BT2, BT} is {0, 0b010x} (address mismatch breakpoint without linking enabled), and AArch32 is not supported at EL1.
- DBGBCR<n>_EL1.MASK is a valid nonzero value and any of the following apply:
  - DBGBCR<n>_EL1.BAS is not 0b1111, and AArch32 is supported at EL0.
  - DBGBVR<n>_EL1[(MASK-1):0] is nonzero.
  - DBGBCR<n>_EL1.{BT2, BT} is {0, 0b0x1x} or {0, 0b1xxx} (Context matching breakpoint).

When any of these conditions apply, the breakpoint behaves as if one of the following:

- DBGBCR<n>_EL1.MASK has been programmed with a defined value, which might be 0b00000 (no mask), other than for a direct read of DBGBCR<n>_EL1.
- The breakpoint is disabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bits [23:20]

Breakpoint Type.

With DBGBCR<n>_EL1.BT2 when implemented, specifies breakpoint type.

BT	Meaning	Applies when
0b0000	Unlinked instruction address match. DBGBVR<n>_EL1 is the address of an instruction.	
0b0001	Linked instruction address match. As 0b0000, but linked to a breakpoint that has linking enabled.	
0b0010	Unlinked Context ID match. If the Effective value of HCR_EL2.E2H is 1, and either the PE is executing at EL0 with HCR_EL2.TGE set to 1 or the PE is executing at EL2, then DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL2. Otherwise, DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL1.	When breakpoint n is context-aware
0b0011	As 0b0010, with linking enabled.	When breakpoint n is context-aware
0b0100	Unlinked instruction address mismatch. DBGBVR<n>_EL1 is the address of an instruction.	When FEAT_BWE is implemented
0b0101	Linked instruction address mismatch. As 0b0100, but linked to a breakpoint that has linking enabled.	When FEAT_BWE is implemented
0b0110	Unlinked CONTEXTIDR_EL1 match. DBGBVR<n>_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1.	When EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b0111	As 0b0110, with linking enabled.	When EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1000	Unlinked VMID match. DBGBVR<n>_EL1.VMID is a VMID compared against VTTBR_EL2.VMID.	When EL2 is implemented and breakpoint n is context-aware
0b1001	As 0b1000, with linking enabled.	When EL2 is implemented and breakpoint n is context-aware
0b1010	Unlinked VMID and Context ID match. DBGBVR<n>_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1, and DBGBVR<n>_EL1.VMID is a VMID compared against VTTBR_EL2.VMID.	When EL2 is implemented and breakpoint n is context-aware
0b1011	As 0b1010, with linking enabled.	When EL2 is implemented and breakpoint n is context-aware
0b1100	Unlinked CONTEXTIDR_EL2 match. DBGBVR<n>_EL1.ContextID2 is a Context ID compared against CONTEXTIDR_EL2.	When EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1101	As 0b1100, with linking enabled.	When EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1110	Unlinked Full Context ID match. DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL1, and DBGBVR<n>_EL1.ContextID2 is compared against CONTEXTIDR_EL2.	When EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1111	As 0b1110, with linking enabled.	When EL2 is implemented,

	FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked Breakpoint Number.

For Linked address matching breakpoints, with DBGBCR<n>_EL1.LBNX when implemented, specifies the index of the breakpoint linked to.

For all other breakpoint types, this field is ignored and reads of the register return an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint n is generated.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

For more information on the effect of programming the fields to a reserved set of values, see 'Reserved DBGBCR<n>_EL1.{SSC, HMC, PMC} values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

For more information, see DBGBCR<n>_EL1.SSC.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [12:9]

Reserved, RES0.

BAS, bits [8:5]

When FEAT_AA32 is implemented:

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state.

The permitted values depend on the breakpoint type.

For Address match breakpoints, the permitted values are:

BAS	Match instruction at	Constraint for debuggers
0b0011	DBGBVR<n>_EL1	Use for T32 instructions.
0b1100	DBGBVR<n>_EL1 + 2	Use for T32 instructions.
0b1111	DBGBVR<n>_EL1	Use for A64 and A32 instructions.

All other values are reserved. For more information, see 'Reserved DBGBCR<n>_EL1.BAS values'.

For more information on using the BAS field in address match breakpoints, see 'Using the BAS field in Address Match breakpoints'.

For Context matching breakpoints, this field is RES1 and ignored.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [4]

Reserved, RES0.

BT2, bit [3]

When FEAT_ABLE is implemented and breakpoint n supports address breakpoint linking:

Breakpoint Type 2. With DBGBCR<n>_EL1.BT, specifies breakpoint type.

BT2	Meaning
0b0	As DBGBCR<n>_EL1.BT.
0b1	As DBGBCR<n>_EL1.BT, but with linking enabled. This value is only defined for the following DBGBCR<n>_EL1.BT values: 0b0000, 0b0001, 0b0100, and 0b0101. All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMC, bits [2:1]

Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

For more information, see DBGBCR<n>_EL1.SSC.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable breakpoint n.

E	Meaning
0b0	Breakpoint n disabled.
0b1	Breakpoint n enabled.

This field is ignored by the PE and treated as zero when all of the following are true:

- Any of the following are true:
  - `HaltOnBreakpointOrWatchpoint()` is FALSE and the Effective value of MDSCR_EL1.EMBWE is 0.
  - `HaltOnBreakpointOrWatchpoint()` is TRUE and the Effective value of EDSCR2.EHBWE is 0.
- FEAT_Debugv8p9 is implemented.
- n >= 16.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBCR<n>_EL1

When FEAT_Debugv8p9 is implemented, a PE is permitted to support up to 64 implemented breakpoints.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGBCR<m>_EL1 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b101

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_BREAKPOINTS) ||
(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16) >=
NUM_BREAKPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.DBGBCRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBCR_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBCR_EL1[m];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBCR_EL1[m];
```

MSR DBGBCR<m>_EL1, <Xt> ; Where m = 0-15

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b101

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_BREAKPOINTS) ||
(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16) >=
NUM_BREAKPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.DBGBCRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBCR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBCR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBCR_EL1[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGBVR<n>_EL1, Debug Breakpoint Value Registers, n = 0 - 63

The DBGBVR<n>_EL1 characteristics are:

Purpose

Holds a virtual address, or a VMID and/or a context ID, for use in breakpoint matching. Forms breakpoint n together with control register DBGBCR<n>_EL1.

Configuration

AArch64 System register DBGBVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGBVR<n>[31:0].

AArch64 System register DBGBVR<n>_EL1 bits [63:32] are architecturally mapped to AArch32 System register DBGBXVR<n>[31:0].

AArch64 System register DBGBVR<n>_EL1 bits [63:0] are architecturally mapped to External register DBGBVR<n>_EL1[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGBVR<n>_EL1 are UNDEFINED.

How this register is interpreted depends on the value of DBGBCR<n>_EL1.BT.

- When DBGBCR<n>_EL1.BT is `0b000x`, this register holds a virtual address.
- When DBGBCR<n>_EL1.BT is `0b001x`, `0b011x`, or `0b110x`, this register holds a Context ID.
- When DBGBCR<n>_EL1.BT is `0b100x`, this register holds a VMID.
- When DBGBCR<n>_EL1.BT is `0b101x`, this register holds a VMID and a Context ID.
- When DBGBCR<n>_EL1.BT is `0b111x`, this register holds two Context ID values.

For other values of DBGBCR<n>_EL1.BT, this register is RES0.

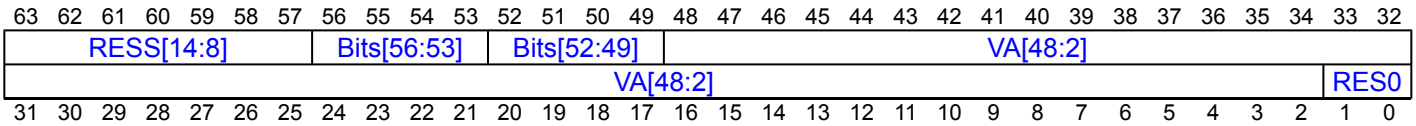
If breakpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGBVR<n>_EL1 is a 64-bit register.

Field descriptions

When DBGBCR<n>_EL1.BT IN {0b000x}:



RESS[14:8], bits [63:57]

Reserved, Sign extended. Software must set all bits in this field to the same value as the most significant bit of the VA field. If all bits in this field are not the same value as the most significant bit of the VA field, then all of the following apply:

- It is CONSTRAINED UNPREDICTABLE whether the PE ignores this field when comparing an address.
- If the breakpoint is not context-aware, it is IMPLEMENTATION DEFINED whether the value read back in each bit of this field is a copy of the most significant bit of the VA field or the value written.

Bits[56:53]

When FEAT_LVA3 is implemented:

VA[56:53], bits [3:0] of bits [56:53]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[7:4], bits [3:0] of bits [56:53]

Extension to RESS[14:8]. For more information, see RESS[14:8].

Bits[52:49]

When FEAT_LVA is implemented:

VA[52:49], bits [3:0] of bits [52:49]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[3:0], bits [3:0] of bits [52:49]

Extension to RESS[14:8]. For more information, see RESS[14:8].

VA[48:2], bits [48:2]

Bits[48:2] of the address value for comparison.

When FEAT_LVA3 is implemented, (VA[56:53]:VA[52:49]) forms the upper part of the address value. If FEAT_LVA3 is not implemented, bits VA[56:53] are part of the RESS field.

When FEAT_LVA is implemented, VA[52:49] forms the upper part of the address value. If FEAT_LVA is not implemented, bits [52:49] are part of the RESS field.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT IN {0b001x}:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
ContextID																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:32]

Reserved, RES0.

ContextID, bits [31:0]

Context ID value for comparison.

The value is compared against CONTEXTIDR_EL2 when the Effective value of HCR_EL2.E2H is 1, and either:

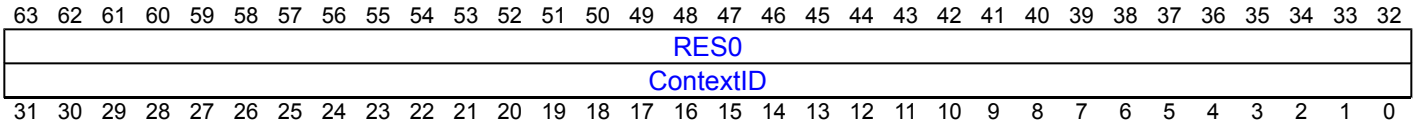
- The PE is executing at EL2.
- HCR_EL2.TGE is 1, the PE is executing at EL0, and EL2 is enabled in the current Security state.

Otherwise, the value is compared against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT IN {0b011x}, EL2 is implemented, and FEAT_Debugv8p1 is implemented:



Bits [63:32]

Reserved, RES0.

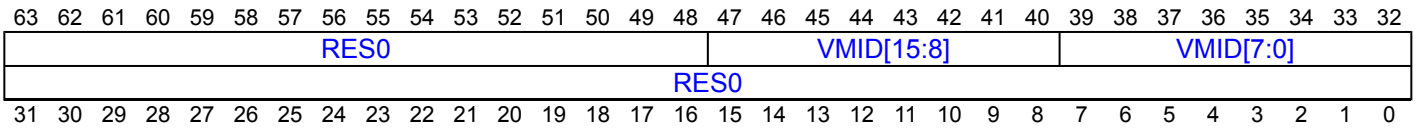
ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT IN {0b100x} and EL2 is implemented:



Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VMID16 is implemented, VTCR_EL2.VS == 1, and EL2 is using AArch64:

Extension to VMID[7:0]. For more information, see DBGBVR<n>_EL1.VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

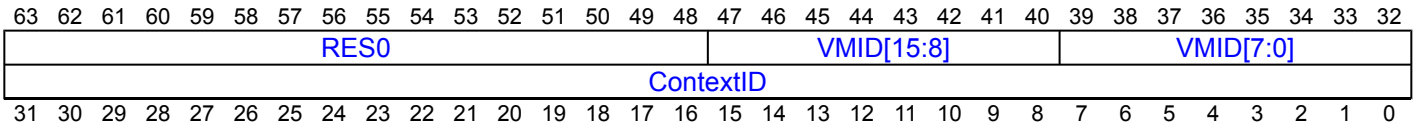
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT IN {0b101x} and EL2 is implemented:



Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VMID16 is implemented, VTCR_EL2.VS == 1, and EL2 is using AArch64:

Extension to VMID[7:0]. For more information, see DBGBVR<n>_EL1.VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

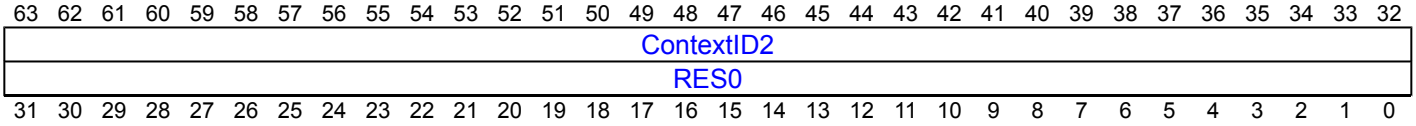
ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT IN {0b110x}, EL2 is implemented, and FEAT_Debugv8p1 is implemented:



ContextID2, bits [63:32]

Context ID value for comparison against CONTEXTIDR_EL2.

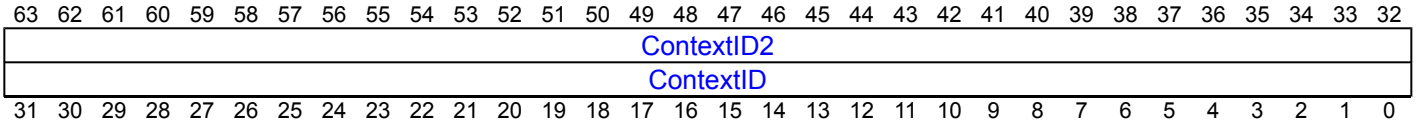
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT IN {0b111x}, EL2 is implemented, and FEAT_Debugv8p1 is implemented:



ContextID2, bits [63:32]

Context ID value for comparison against CONTEXTIDR_EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBVR<n>_EL1

When FEAT_Debugv8p9 is implemented, a PE is permitted to support up to 64 implemented breakpoints.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGBVR<m>_EL1 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b100

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_BREAKPOINTS) ||
(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16) >=
NUM_BREAKPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.DBGBVRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBVR_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBVR_EL1[m];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBVR_EL1[m];
```

```
MSR DBGBVR<m>_EL1, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b100

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_BREAKPOINTS) ||
(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16) >=
NUM_BREAKPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.DBGBVRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBVR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBVR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBVR_EL1[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMCLR_EL1, Debug CLAIM Tag Clear Register

The DBGCLAIMCLR_EL1 characteristics are:

Purpose

Used by software to read the values of the CLAIM tag bits, and to clear CLAIM tag bits to 0.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMSET_EL1 register.

Configuration

AArch64 System register DBGCLAIMCLR_EL1 bits [63:0] are architecturally mapped to AArch64 System register DBGCLAIMSET_EL1[63:0].

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMCLR[31:0].

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMSET[31:0].

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMCLR_EL1[31:0].

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMSET_EL1[31:0].

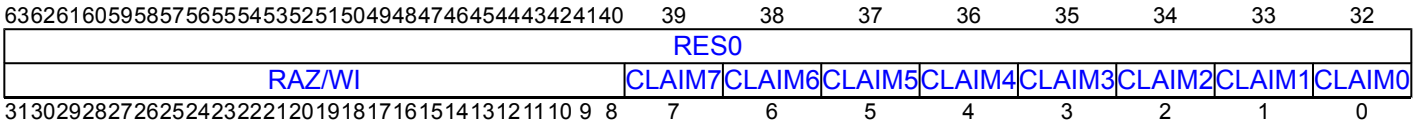
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGCLAIMCLR_EL1 are UNDEFINED.

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMCLR_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

Bits [31:8]

Reserved, RAZ/WI.

CLAIM<m>, bit [m], for m = 7 to 0

Claim Tag Clear. Indicates the current status of Claim Tag bit <m>, and is used to clear Claim Tag bit <m> to 0.

CLAIM<m>	Meaning
0b0	On a read: Claim Tag bit <m> is not set. On a write: Ignored.
0b1	On a read: Claim Tag bit <m> is set. On a write: Clear Claim tag bit <m> to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Access to this field is **W1C**.

Accessing DBGCLAIMCLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGCLAIMCLR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0111	0b1001	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.DBGCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGCLAIMCLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGCLAIMCLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DBGCLAIMCLR_EL1;
```

```
MSR DBGCLAIMCLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0111	0b1001	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.DBGCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGCLAIMCLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGCLAIMCLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DBGCLAIMCLR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMSET_EL1, Debug CLAIM Tag Set Register

The DBGCLAIMSET_EL1 characteristics are:

Purpose

Used by software to set the CLAIM tag bits to 1.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMCLR_EL1 register.

Configuration

AArch64 System register DBGCLAIMSET_EL1 bits [63:0] are architecturally mapped to AArch64 System register DBGCLAIMCLR_EL1[63:0].

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMSET[31:0].

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMCLR[31:0].

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMSET_EL1[31:0].

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMCLR_EL1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGCLAIMSET_EL1 are UNDEFINED.

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMSET_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
RAZ/WI																								CLAIM7	CLAIM6	CLAIM5	CLAIM4	CLAIM3	CLAIM2	CLAIM1	CLAIM0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:32]

Reserved, RES0.

Bits [31:8]

Reserved, RAZ/WI.

CLAIM<m>, bit [m], for m = 7 to 0

Claim Tag Set. Used to set Claim Tag bit <m> to 1.

CLAIM<m>	Meaning
0b0	On a write: Ignored.
0b1	On a write: Set Claim Tag bit <m> to 1.

Access to this field is **RAO/W1S**.

Accessing DBGCLAIMSET_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGCLAIMSET_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0111	0b1000	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.DBGCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGCLAIMSET_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGCLAIMSET_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DBGCLAIMSET_EL1;
```

```
MSR DBGCLAIMSET_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0111	0b1000	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.DBGCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGCLAIMSET_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGCLAIMSET_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DBGCLAIMSET_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTR_EL0, Debug Data Transfer Register, half-duplex

The DBGDTR_EL0 characteristics are:

Purpose

Transfers 64 bits of data between the PE and an external debugger. Can transfer both ways using only a single register.

Configuration

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch32 System register DBGDTRRXint[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to External register DBGDTRRX_EL0[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch64 System register DBGDTRRX_EL0[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXint[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to External register DBGDTRTX_EL0[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch64 System register DBGDTRTX_EL0[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch32 System register DBGDTRTXint[31:0] when read.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to External register DBGDTRTX_EL0[31:0] when read.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch64 System register DBGDTRTX_EL0[31:0] when read.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRRXint[31:0] when read.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to External register DBGDTRRX_EL0[31:0] when read.

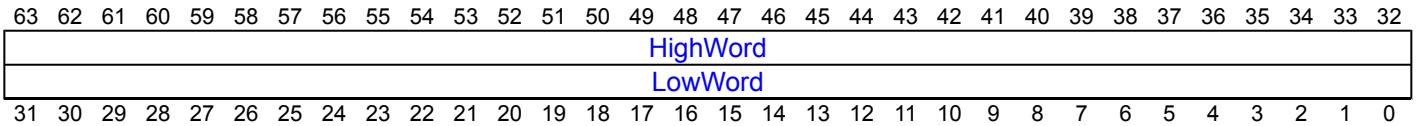
AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch64 System register DBGDTRRX_EL0[31:0] when read.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGDTR_EL0 are UNDEFINED.

Attributes

DBGDTR_EL0 is a 64-bit register.

Field descriptions



HighWord, bits [63:32]

Writes to this register set DTRRX to the value in this field and do not change RXfull.

Reads of this register:

- If RXfull is 1, return the last value written to DTRTX.
- If RXfull is 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

LowWord, bits [31:0]

Writes to this register set DTRTX to the value in this field and set TXfull to 1.

Reads of this register:

- If RXfull is 1, return the last value written to DTRRX.
- If RXfull is 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

Accessing DBGDTR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGDTR_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b0000	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif Halted() then
    X[t, 64] = Read_DBGDTR_EL0(64);
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = Read_DBGDTR_EL0(64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = Read_DBGDTR_EL0(64);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = Read_DBGDTR_EL0(64);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Read_DBGDTR_EL0(64);
```

```
MSR DBGDTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b0000	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif Halted() then
    Write_DBGDTR_EL0(X[t, 64]);
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 64]);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 64]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 64]);
elsif PSTATE.EL == EL3 then
    Write_DBGDTR_EL0(X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRRX_EL0, Debug Data Transfer Register, Receive

The DBGDTRRX_EL0 characteristics are:

Purpose

Transfers data from an external debugger to the PE. For example, it is used by a debugger transferring commands and data to a debug target. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communications Channel.

Configuration

AArch64 System register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRRXint[31:0].

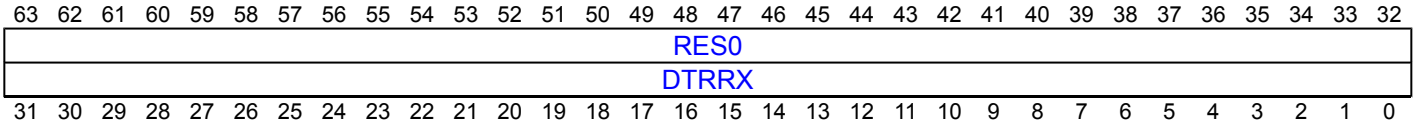
AArch64 System register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to External register DBGDTRRX_EL0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGDTRRX_EL0 are UNDEFINED.

Attributes

DBGDTRRX_EL0 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

DTRRX, bits [31:0]

DTRRX. Reads of this register:

- If RXfull is 1, return the last value written to DTRRX.
- If RXfull is 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRRX_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGDTRRX_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b0000	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif Halted() then
    X[t, 32] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 32] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 32] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 32] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL3 then
    X[t, 32] = Read_DBGDTR_EL0(32);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRTX_EL0, Debug Data Transfer Register, Transmit

The DBGDTRTX_EL0 characteristics are:

Purpose

Transfers data from the PE to an external debugger. For example, it is used by a debug target to transfer data to the debugger. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communication Channel.

Configuration

AArch64 System register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXint[31:0].

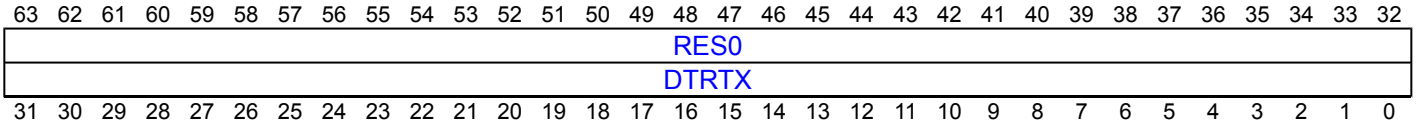
AArch64 System register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to External register DBGDTRTX_EL0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGDTRTX_EL0 are UNDEFINED.

Attributes

DBGDTRTX_EL0 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

DTRTX, bits [31:0]

DTRTX. Writes to this register:

- If TXfull is 1, DTRTX is set to an UNKNOWN value.
- If TXfull is 0, update the value in DTRTX.

After the write, TXfull is set to 1.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRTX_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MSR DBGDTRTX_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b0000	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif Halted() then
    Write_DBGDTR_EL0(X[t, 32]);
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 32]);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 32]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 32]);
elsif PSTATE.EL == EL3 then
    Write_DBGDTR_EL0(X[t, 32]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGPRCR_EL1, Debug Power Control Register

The DBGPRCR_EL1 characteristics are:

Purpose

Controls behavior of the PE on powerdown request.

Configuration

AArch64 System register DBGPRCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGPRCR[31:0].

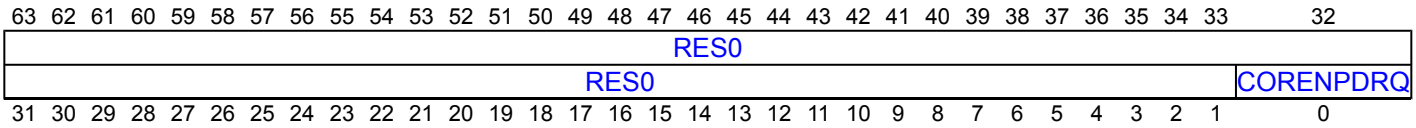
AArch64 System register DBGPRCR_EL1 bit [0] is architecturally mapped to External register EDPRCR[0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGPRCR_EL1 are UNDEFINED.

Attributes

DBGPRCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

CORENPDRQ, bit [0]

When FEAT_DoPD is implemented:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

CORENPDRQ	Meaning
0b0	If the system responds to a powerdown request, it powers down Core power domain.
0b1	If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGNOPWRDWN** signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to its Cold reset value on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

On a Cold reset, if the powerup request is implemented and the powerup request has been asserted, this field is set to an IMPLEMENTATION DEFINED choice of 0 or 1. If the powerup request is not asserted, this field is set to 0.

Otherwise:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

CORENPDRQ	Meaning
0b0	If the system responds to a powerdown request, it powers down Core power domain.
0b1	If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGNOPWRDWN** signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the value of EDPRCR.COREPURQ on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The reset behavior of this field is:

- On a Cold reset, this field resets to the value in EDPRCR.COREPURQ.

Accessing DBGPRCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGPRCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.DBGPRCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGPRCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGPRCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DBGPRCR_EL1;
```

```
MSR DBGPRCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.DBGPRCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGPRCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGPRCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DBGPRCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGVCR32_EL2, Debug Vector Catch Register

The DBGVCR32_EL2 characteristics are:

Purpose

Allows access to the AArch32 register DBGVCR from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register DBGVCR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register DBGVCR[31:0].

This register is present only when FEAT_AA32EL1 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DBGVCR32_EL2 are UNDEFINED.

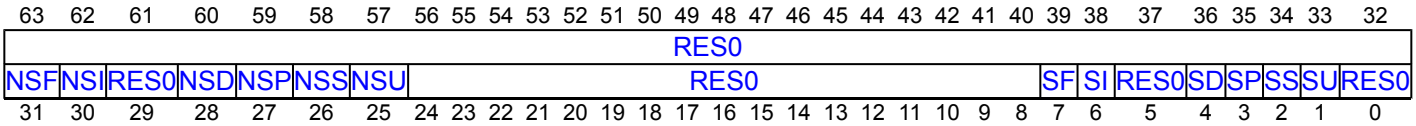
If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

Attributes

DBGVCR32_EL2 is a 64-bit register.

Field descriptions

When EL3 is implemented:



Bits [63:32]

Reserved, RES0.

NSF, bit [31]

FIQ vector catch enable in Non-secure state.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSI, bit [30]

IRQ vector catch enable in Non-secure state.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

NSD, bit [28]

Data Abort exception vector catch enable in Non-secure state.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSP, bit [27]

Prefetch Abort vector catch enable in Non-secure state.

The exception vector offset is `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSS, bit [26]

Supervisor Call (SVC) vector catch enable in Non-secure state.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSU, bit [25]

Undefined Instruction vector catch enable in Non-secure state.

The exception vector offset is `0x04`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [24:8]

Reserved, RES0.

SF, bit [7]

FIQ vector catch enable in Secure state.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SI, bit [6]

IRQ vector catch enable in Secure state.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

SD, bit [4]

Data Abort exception vector catch enable in Secure state.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SP, bit [3]

Prefetch Abort vector catch enable in Secure state.

The exception vector offset is `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SS, bit [2]

Supervisor Call (SVC) vector catch enable in Secure state.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SU, bit [1]

Undefined Instruction vector catch enable in Secure state.

The exception vector offset is `0x04`.

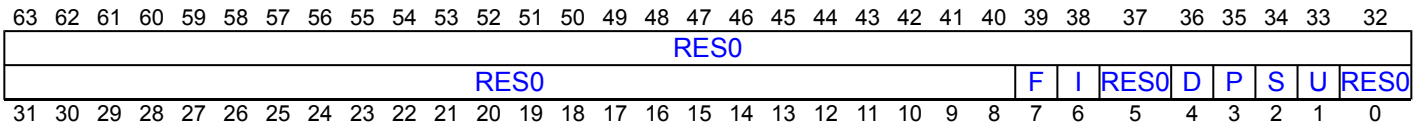
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

When EL3 is not implemented:



Bits [63:8]

Reserved, RES0.

F, bit [7]

FIQ vector catch enable.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [6]

IRQ vector catch enable.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

D, bit [4]

Data Abort exception vector catch enable.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P, bit [3]

Prefetch Abort vector catch enable.

The exception vector offset `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [2]

Supervisor Call (SVC) vector catch enable.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

U, bit [1]

Undefined Instruction vector catch enable.

The exception vector offset is `0x04`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

Accessing DBGVCR32_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGVCR32_EL2
```

op0	op1	CRn	CRm	op2
0b10	0b100	0b0000	0b0111	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGVCR32_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DBGVCR32_EL2;
```

```
MSR DBGVCR32_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b100	0b0000	0b0111	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGVCR32_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DBGVCR32_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWCR<n>_EL1, Debug Watchpoint Control Registers, n = 0 - 63

The DBGWCR<n>_EL1 characteristics are:

Purpose

Holds control information for a watchpoint. Forms watchpoint n together with value register DBGWVR<n>_EL1.

Configuration

AArch64 System register DBGWCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGWCR<n>[31:0].

AArch64 System register DBGWCR<n>_EL1 bits [31:0] are architecturally mapped to External register DBGWCR<n>_EL1[31:0].

AArch64 System register DBGWCR<n>_EL1 bits [63:32] are architecturally mapped to External register DBGWCR<n>_EL1[63:32] when FEAT_Debugv8p9 is implemented.

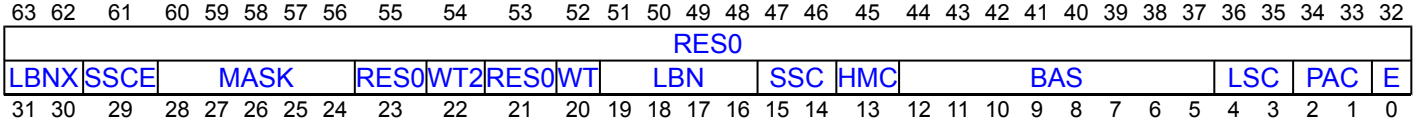
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGWCR<n>_EL1 are UNDEFINED.

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGWCR<n>_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

LBNX, bits [31:30]

When FEAT_Debugv8p9 is implemented:

Linked Breakpoint Number.

For Linked data address watchpoints, with DBGWCR<n>_EL1.LBN, specifies the index of the breakpoint linked to.

For all other watchpoint types, this field is ignored and reads of the register return an UNKNOWN value.

This field extends DBGWCR<n>_EL1.LBN to support up to 64 implemented breakpoints.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSCE, bit [29]

When FEAT_RME is implemented:

Security State Control Extended.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MASK, bits [28:24]

Address Mask. Only address ranges up to 2GB can be watched using a single mask.

MASK	Meaning
0b00000	No mask.
0b00011..0b11111	Number of address bits masked.

All other values are reserved.

Indicates the number of masked address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b11111 masking 31 address bits (0x7FFFFFFF mask for address).

If programmed with a reserved value, the watchpoint behaves as if one of the following:

- DBGWCR<n>_EL1.MASK has been programmed with a defined value, which might be 0b00000 (no mask), other than for a direct read of DBGWCR<n>_EL1.
- The watchpoint is disabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [23]

Reserved, RES0.

WT2, bit [22]

When FEAT_BWE2 is implemented:

Watchpoint Type 2. With DBGWCR<n>_EL1.WT, specifies watchpoint type.

WT2	Meaning
0b0	Watchpoint n is an address match watchpoint.
0b1	Watchpoint n is an address mismatch watchpoint.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [21]

Reserved, RES0.

WT, bit [20]

Watchpoint type. Possible values are:

WT	Meaning
0b0	Unlinked watchpoint.
0b1	Linked watchpoint.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked Breakpoint Number.

For Linked data address watchpoints, with DBGWCR<n>_EL1.LBNX when implemented, specifies the index of the breakpoint linked to.

For all other watchpoint types, this field is ignored and reads of the register return an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

For more information on the effect of programming the fields to a reserved value, see 'Reserved DBGWCR<n>_EL1.{SSC, HMC, PAC} values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

BAS, bits [12:5]

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by DBGWVR<n>_EL1 is being watched.

BAS	Description
xxxxxxx1	Match byte at DBGWVR<n>_EL1
xxxxxx1x	Match byte at DBGWVR<n>_EL1 + 1
xxxxx1xx	Match byte at DBGWVR<n>_EL1 + 2
xxxx1xxx	Match byte at DBGWVR<n>_EL1 + 3

In cases where DBGWVR<n>_EL1 addresses a double-word:

BAS	Description, if DBGWVR<n>_EL1[2] == 0
xxx1xxxx	Match byte at DBGWVR<n>_EL1 + 4
xx1xxxxx	Match byte at DBGWVR<n>_EL1 + 5
x1xxxxxx	Match byte at DBGWVR<n>_EL1 + 6
1xxxxxxx	Match byte at DBGWVR<n>_EL1 + 7

If DBGWVR<n>_EL1[2] == 1, only BAS[3:0] are used and BAS[7:4] are ignored. Arm deprecates setting DBGWVR<n>_EL1[2] == 1.

The valid values for BAS are nonzero binary numbers all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<n>_EL1.BAS values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LSC, bits [4:3]

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

LSC	Meaning
0b01	Match instructions that load from a watchpointed address.
0b10	Match instructions that store to a watchpointed address.
0b11	Match instructions that load from or store to a watchpointed address.

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PAC, bits [2:1]

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint n is generated.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable watchpoint n.

E	Meaning
0b0	Watchpoint n disabled.
0b1	Watchpoint n enabled.

This field is ignored by the PE and treated as zero when all of the following are true:

- Any of the following are true:
  - `HaltOnBreakpointOrWatchpoint()` is FALSE and the Effective value of MDSCR_EL1.EMBWE is 0.
  - `HaltOnBreakpointOrWatchpoint()` is TRUE and the Effective value of EDSCR2.EHBWE is 0.
- FEAT_Debugv8p9 is implemented.
- n >= 16.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGWCR<n>_EL1

When FEAT_Debugv8p9 is implemented, a PE is permitted to support up to 64 implemented watchpoints.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGWCR<m>_EL1 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b111

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_WATCHPOINTS) ||
(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(MDSELR_EL1.BANK) * 16) >= NUM_WATCHPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.DBGWCRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWCR_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWCR_EL1[m];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWCR_EL1[m];
```

```
MSR DBGWCR<m>_EL1, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b111

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_WATCHPOINTS) ||
(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(MDSELR_EL1.BANK) * 16) >= NUM_WATCHPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.DBGWCRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWCR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWCR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWCR_EL1[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWVR<n>_EL1, Debug Watchpoint Value Registers, n = 0 - 63

The DBGWVR<n>_EL1 characteristics are:

Purpose

Holds a data address value for use in watchpoint matching. Forms watchpoint n together with control register DBGWCR<n>_EL1.

Configuration

AArch64 System register DBGWVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGWVR<n>[31:0].

AArch64 System register DBGWVR<n>_EL1 bits [63:0] are architecturally mapped to External register DBGWVR<n>_EL1[63:0].

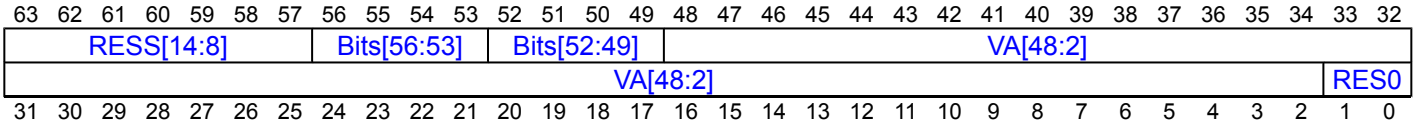
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGWVR<n>_EL1 are UNDEFINED.

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGWVR<n>_EL1 is a 64-bit register.

Field descriptions



RESS[14:8], bits [63:57]

Reserved, Sign extended. Software must set all bits in this field to the same value as the most significant bit of the VA field. If all bits in this field are not the same value as the most significant bit of the VA field, then all of the following apply:

- It is CONSTRAINED UNPREDICTABLE whether the PE ignores this field when comparing an address.
- It is IMPLEMENTATION DEFINED whether the value read back in each bit of this field is a copy of the most significant bit of the VA field or the value written.

Bits[56:53]

When FEAT_LVA3 is implemented:

VA[56:53], bits [3:0] of bits [56:53]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[7:4], bits [3:0] of bits [56:53]

Extension to RESS[14:8]. For more information, see RESS[14:8].

Bits[52:49]

When FEAT_LVA is implemented:

VA[52:49], bits [3:0] of bits [52:49]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[3:0], bits [3:0] of bits [52:49]

Extension to RESS[14:8]. For more information, see RESS[14:8].

VA[48:2], bits [48:2]

Bits[48:2] of the address value for comparison.

When FEAT_LVA3 is implemented, (VA[56:53]:VA[52:49]) forms the upper part of the address value. If FEAT_LVA3 is not implemented, bits VA[56:53] are part of the RESS field.

When FEAT_LVA is implemented, VA[52:49] forms the upper part of the address value. If FEAT_LVA is not implemented, bits [52:49] are part of the RESS field.

Arm deprecates setting DBGWVR<n>_EL1[2] == 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

Accessing DBGWVR<n>_EL1

When FEAT_Debugv8p9 is implemented, a PE is permitted to support up to 64 implemented watchpoints.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGWVR<m>_EL1 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b110

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_WATCHPOINTS) ||
(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(MDSELR_EL1.BANK) * 16) >= NUM_WATCHPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.DBGWVRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWVR_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWVR_EL1[m];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWVR_EL1[m];
```

```
MSR DBGWVR<m>_EL1, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b110

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_WATCHPOINTS) ||
(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(MDSELR_EL1.BANK) * 16) >= NUM_WATCHPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.DBGWVRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWVR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWVR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWVR_EL1[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGDSW, Clean of Data and Allocation Tags by Set/Way

The DC CGDSW characteristics are:

Purpose

Clean data and Allocation Tags in data cache by set/way.

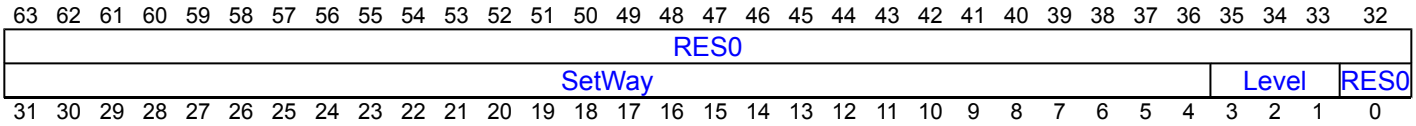
Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC CGDSW are UNDEFINED.

Attributes

DC CGDSW is a 64-bit System instruction.

Field descriptions



Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DC CGDSW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:

- No cache lines.
- A single arbitrary cache line.
- Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CGDSW, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1010	0b110

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGDVAC, Clean of Data and Allocation Tags by VA to PoC

The DC CGDVAC characteristics are:

Purpose

Clean data and Allocation Tags in data cache by address to Point of Coherency.

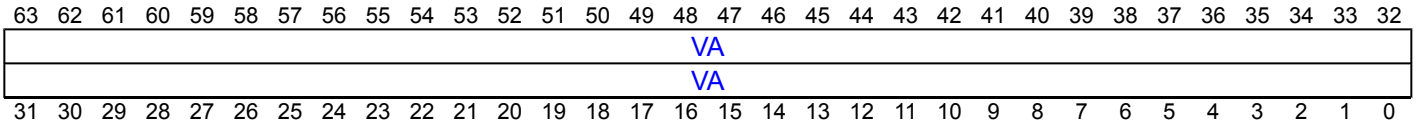
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CGDVAC are UNDEFINED.

Attributes

DC CGDVAC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CGDVAC

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CGDVAC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1010	0b101

```
if !IsFeatureImplemented(FEAT_MTE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGDVADP, Clean of Data and Allocation Tags by VA to PoDP

The DC CGDVADP characteristics are:

Purpose

Clean Allocation Tags and data in data cache by address to Point of Deep Persistence.

If the memory system does not identify a Point of Deep Persistence, then this instruction behaves as a DC CGDVAP.

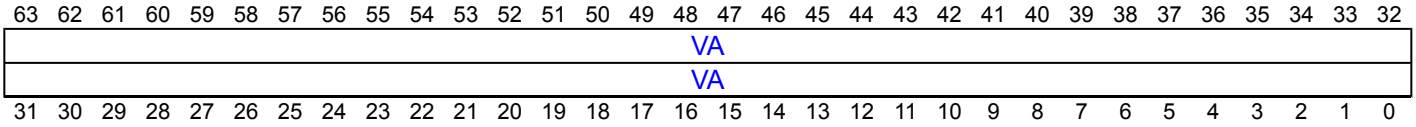
Configuration

This instruction is present only when FEAT_DPB2 is implemented and FEAT_MTE is implemented. Otherwise, direct accesses to DC CGDVADP are UNDEFINED.

Attributes

DC CGDVADP is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CGDVADP

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CGDVADP, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1101	0b101

```
if !(IsFeatureImplemented(FEAT_DPB2) && IsFeatureImplemented(FEAT_MTE)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoDP);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVADP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoDP);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoDP);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoDP);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGDVAOC, Clean of Data and Allocation Tags by VA to Outer Cache level

The DC CGDVAOC characteristics are:

Purpose

Clean data and Allocation Tags in data cache by address to Outer cache level.

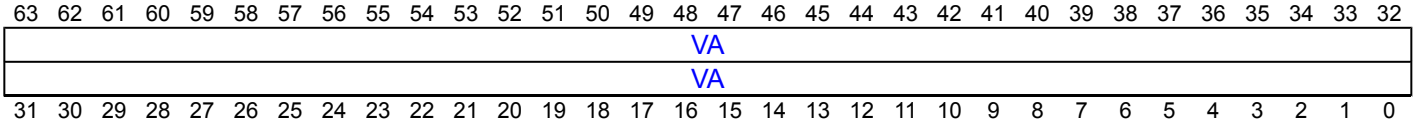
Configuration

This instruction is present only when FEAT_OCCMO is implemented, FEAT_MTE is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CGDVAOC are UNDEFINED.

Attributes

DC CGDVAOC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CGDVAOC

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CGDVAOC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1011	0b111

```
if !(IsFeatureImplemented(FEAT_OCCMO) && IsFeatureImplemented(FEAT_MTE) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_OuterCache);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGDVAP, Clean of Data and Allocation Tags by VA to PoP

The DC CGDVAP characteristics are:

Purpose

Clean data and Allocation Tags in data cache by address to Point of Persistence.

If the memory system does not identify a Point of Persistence, then this instruction behaves as a DC CGDVAC.

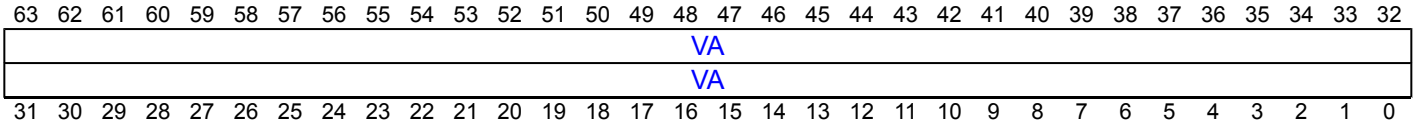
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CGDVAP are UNDEFINED.

Attributes

DC CGDVAP is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CGDVAP

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CGDVAP, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1100	0b101

```
if !IsFeatureImplemented(FEAT_MTE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoP);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVAP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoP);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoP);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoP);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGSW, Clean of Allocation Tags by Set/Way

The DC CGSW characteristics are:

Purpose

Clean Allocation Tags in data cache by set/way.

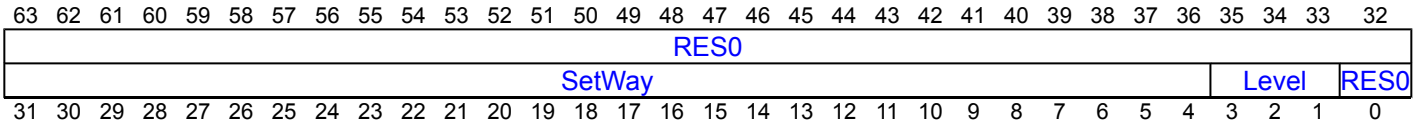
Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC CGSW are UNDEFINED.

Attributes

DC CGSW is a 64-bit System instruction.

Field descriptions



Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DC CGSW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:

- No cache lines.
- A single arbitrary cache line.
- Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CGSW, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1010	0b100

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGVAC, Clean of Allocation Tags by VA to PoC

The DC CGVAC characteristics are:

Purpose

Clean Allocation Tags in data cache by address to Point of Coherency.

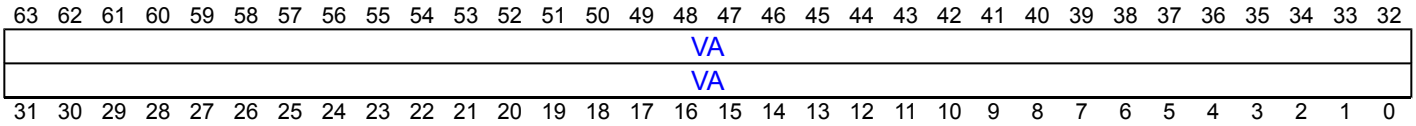
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CGVAC are UNDEFINED.

Attributes

DC CGVAC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CGVAC

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CGVAC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_MTE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGVADP, Clean of Allocation Tags by VA to PoDP

The DC CGVADP characteristics are:

Purpose

Clean Allocation tags by address to Point of Deep Persistence.

If the memory system does not identify a Point of Deep Persistence, then this instruction behaves as a DC CGVAP.

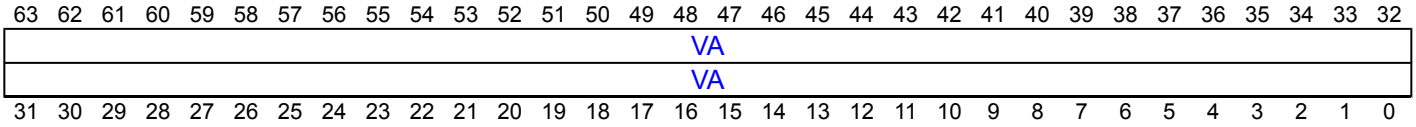
Configuration

This instruction is present only when FEAT_DPB2 is implemented and FEAT_MTE is implemented. Otherwise, direct accesses to DC CGVADP are UNDEFINED.

Attributes

DC CGVADP is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CGVADP

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CGVADP, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_DPB2) && IsFeatureImplemented(FEAT_MTE)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoDP);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVADP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoDP);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoDP);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoDP);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGVAP, Clean of Allocation Tags by VA to PoP

The DC CGVAP characteristics are:

Purpose

Clean Allocation Tags in data cache by address to Point of Persistence.

If the memory system does not identify a Point of Persistence, then this instruction behaves as a DC CGVAC.

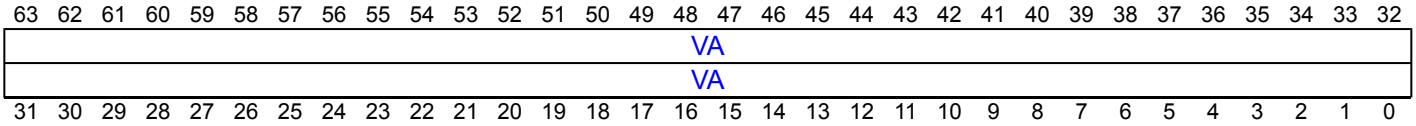
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CGVAP are UNDEFINED.

Attributes

DC CGVAP is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CGVAP

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CGVAP, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1100	0b011

```
if !IsFeatureImplemented(FEAT_MTE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoP);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVAP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoP);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoP);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoP);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGDPAE, Clean and invalidate of data and allocation tags by PA to PoE

The DC CIGDPAE characteristics are:

Purpose

Clean and invalidate of data and allocation tags by PA to PoE.

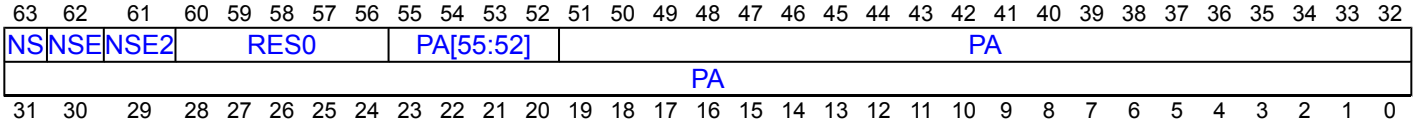
Configuration

This instruction is present only when FEAT_MEC is implemented, FEAT_MTE2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CIGDPAE are UNDEFINED.

Attributes

DC CIGDPAE is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME_GDI is implemented:

Together with the NSE2 and NSE field, this field specifies the target physical address space.

NSE2	NSE	NS	Meaning
0b0	0b0	0b0	Reserved.
0b0	0b0	0b1	Reserved.
0b0	0b1	0b0	Reserved.
0b0	0b1	0b1	Realm.
0b1	0b0	0b0	System Agent.
0b1	0b0	0b1	NS Protected.
0b1	0b1	0b0	Reserved.
0b1	0b1	0b1	Reserved.

If {NSE2, NSE, NS} is reserved, then no cache entries are required to be cleaned or invalidated.

Otherwise:

Together with the NSE field, this field specifies the target physical address space.

NSE	NS	Meaning
0b0	0b0	Reserved.
0b0	0b1	Reserved.
0b1	0b0	Reserved.
0b1	0b1	Realm.

If {NSE, NS} is not {1, 1}, then no cache entries are required to be cleaned or invalidated.

NSE, bit [62]

If FEAT_RME_GDI is implemented, this field together with the NS and NSE2 fields, specifies the target physical address space.

Otherwise, this field and the NS field specify the physical address space

For a description of the values derived by evaluating NS, NSE, and NSE2 together, see DC CIGDPAE.NS.

NSE2, bit [61]

When FEAT_RME_GDI is implemented:

Together with the NS and NSE field, this field specifies the target physical address space.

For a description of the values derived by evaluating NS and NSE together, see DC CIGDPAE.NS.

Otherwise:

Reserved, RES0.

Bits [60:56]

Reserved, RES0.

PA[55:52], bits [55:52]

When FEAT_D128 is implemented:

Extension to PA[51:0] if ID_AA64MMFR0_EL1.PARange = 0111. For more information, see PA[51:0].

Otherwise:

Reserved, RES0.

PA, bits [51:0]

Physical address to use. No alignment restrictions apply to this PA.

Executing DC CIGDPAE

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIGDPAE, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b0111	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_MTE2) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoE);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoE);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGDPAPA, Clean and Invalidate of Data and Allocation Tags by PA to PoPA

The DC CIGDPAPA characteristics are:

Purpose

Clean and Invalidate data and Allocation Tags in data cache by physical address to the Point of Physical Aliasing.

Note

This instruction cleans and invalidates all copies of the Location specified in the Xt argument, irrespective of any MECID associated with the Location. Memory accesses resulting from the Clean operation use the MECID associated with the cache entry.

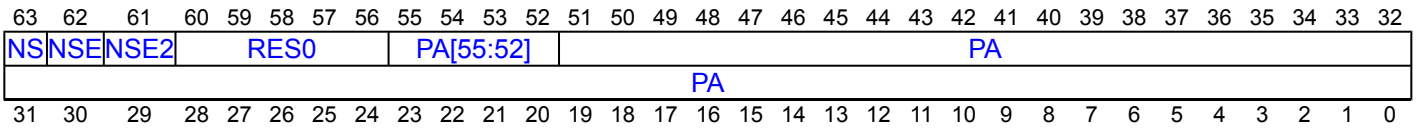
Configuration

This instruction is present only when FEAT_RME is implemented, FEAT_MTE2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CIGDPAPA are UNDEFINED.

Attributes

DC CIGDPAPA is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME_GDI is implemented:

Together with the NSE2 and NSE field, this field specifies the target physical address space.

NSE2	NSE	NS	Meaning
0b0	0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b0	0b1	Non-secure.
0b0	0b1	0b0	Root.
0b0	0b1	0b1	Realm.
0b1	0b0	0b0	System Agent.
0b1	0b0	0b1	NS Protected.
0b1	0b1	0b0	Reserved.
0b1	0b1	0b1	Reserved.

If {NSE2, NSE, NS} is reserved, then no cache entries are required to be cleaned or invalidated.

Otherwise:

Together with the NSE field, this field specifies the target physical address space.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

If FEAT_SEL2 is not implemented, and {NSE, NS} == {0b0, 0b0}, then no cache entries are required to be cleaned or invalidated

NSE, bit [62]

If FEAT_RME_GDI is implemented, this field together with the NS and NSE2 fields, specifies the target physical address space.

Otherwise, this field and the NS field specify the physical address space

For a description of the values derived by evaluating NS, NSE, and NSE2 together, see DC CIGDPAPA.NS.

NSE2, bit [61]

When FEAT_RME_GDI is implemented:

Together with the NS and NSE field, this field specifies the target physical address space.

For a description of the values derived by evaluating NS and NSE together, see DC CIGDPAPA.NS.

Otherwise:

Reserved, RES0.

Bits [60:56]

Reserved, RES0.

PA[55:52], bits [55:52]

When FEAT_D128 is implemented:

Extension to PA[51:0] if ID_AA64MMFR0_EL1.PARange = 0111. For more information see PA[51:0].

Otherwise:

Reserved, RES0.

PA, bits [51:0]

Physical address to use. No alignment restrictions apply to this PA.

Executing DC CIGDPAPA

- This instruction is not subject to any translation, permission checks, or granule protection checks.
- This instruction affects all caches in the Outer Shareable shareability domain.
- This instruction has the same ordering, observability, and completion behavior as VA-based cache maintenance instructions issued to the Outer Shareable shareability domain.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CIGDPAPA, <Xt>

op0	op1	CRn	CRm	op2
0b01	0b110	0b0111	0b1110	0b101

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_MTE2) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoPA);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGDSW, Clean and Invalidate of Data and Allocation Tags by Set/Way

The DC CIGDSW characteristics are:

Purpose

Clean and Invalidate data and Allocation Tags in data cache by set/way.

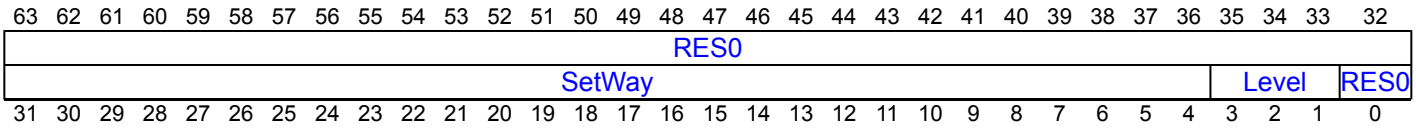
Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC CIGDSW are UNDEFINED.

Attributes

DC CIGDSW is a 64-bit System instruction.

Field descriptions



Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DC CIGDSW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
    - No cache lines.
    - A single arbitrary cache line.
    - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIGDSW, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1110	0b110

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGDVAC, Clean and Invalidate of Data and Allocation Tags by VA to PoC

The DC CIGDVAC characteristics are:

Purpose

Clean and Invalidate data and Allocation Tags in data cache by address to Point of Coherency.

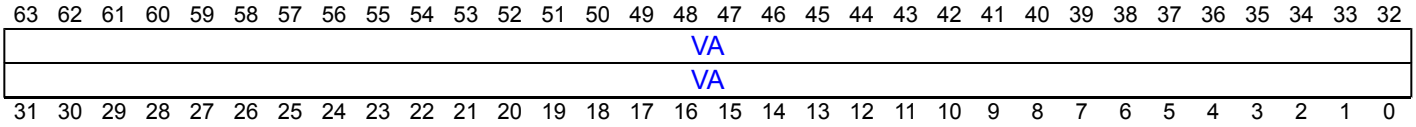
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CIGDVAC are UNDEFINED.

Attributes

DC CIGDVAC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CIGDVAC

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIGDVAC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1110	0b101

```
if !IsFeatureImplemented(FEAT_MTE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGDVAOC, Clean and Invalidate of Data and Allocation Tags by VA to Outer Cache level

The DC CIGDVAOC characteristics are:

Purpose

Clean and invalidate of data and Allocation Tags in data cache by address to Outer cache level.

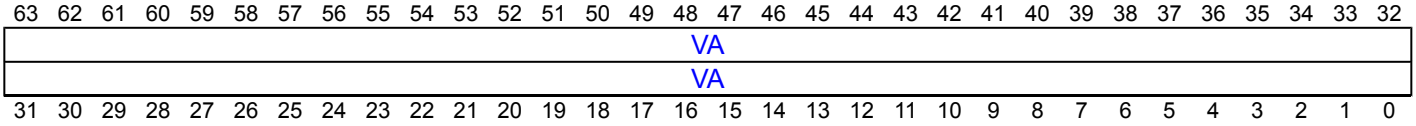
Configuration

This instruction is present only when FEAT_OCCMO is implemented, FEAT_MTE is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CIGDVAOC are UNDEFINED.

Attributes

DC CIGDVAOC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CIGDVAOC

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIGDVAOC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1111	0b111

```
if !(IsFeatureImplemented(FEAT_OCCMO) && IsFeatureImplemented(FEAT_MTE) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_OuterCache);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGDVAPS, Clean and Invalidate of Data and Allocation Tags by VA to PoPS

The DC CIGDVAPS characteristics are:

Purpose

Clean and Invalidate data and Allocation Tags in data cache by virtual address to the Point of Physical Storage.

Note

This instruction cleans and invalidates all copies of the Location specified in the Xt argument, irrespective of any MECID associated with the Location. Memory accesses resulting from the Clean operation use the MECID associated with the cache entry.

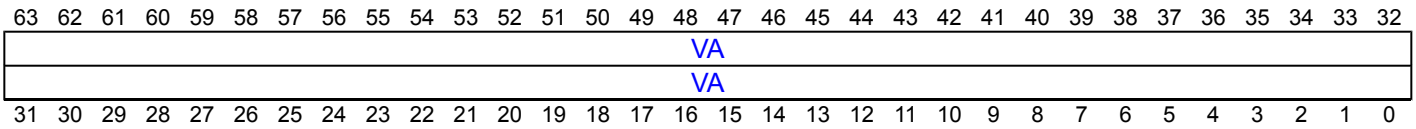
Configuration

This instruction is present only when FEAT_PoPS is implemented, FEAT_MTE2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CIGDVAPS are UNDEFINED.

Attributes

DC CIGDVAPS is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CIGDVAPS

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIGDVAPS, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1111	0b101

```
if !(IsFeatureImplemented(FEAT_PoPS) && IsFeatureImplemented(FEAT_MTE2) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGITR2_EL2.nDCCIVAPS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoPS);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoPS);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoPS);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGSW, Clean and Invalidate of Allocation Tags by Set/Way

The DC CIGSW characteristics are:

Purpose

Clean and Invalidate Allocation Tags in data cache by set/way.

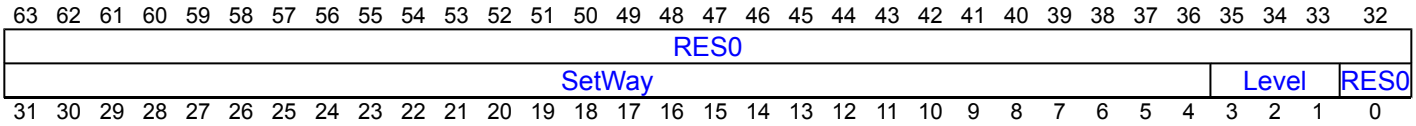
Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC CIGSW are UNDEFINED.

Attributes

DC CIGSW is a 64-bit System instruction.

Field descriptions



Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DC CIGSW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:

- No cache lines.
- A single arbitrary cache line.
- Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIGSW, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1110	0b100

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGVAC, Clean and Invalidate of Allocation Tags by VA to PoC

The DC CIGVAC characteristics are:

Purpose

Clean and Invalidate Allocation Tags in data cache by address to Point of Coherency.

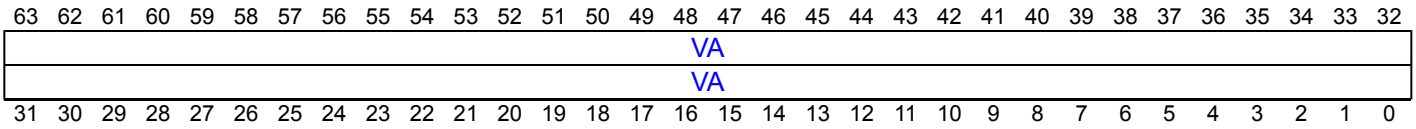
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CIGVAC are UNDEFINED.

Attributes

DC CIGVAC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CIGVAC

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIGVAC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1110	0b011

```
if !IsFeatureImplemented(FEAT_MTE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIPAE, Data or unified Cache line Clean and Invalidate by PA to PoE

The DC CIPAE characteristics are:

Purpose

Data or unified Cache line Clean and Invalidate by PA to PoE.

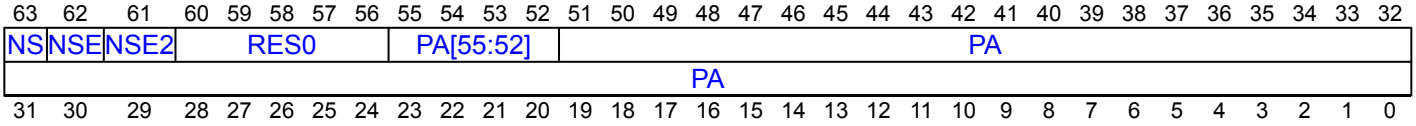
Configuration

This instruction is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CIPAE are UNDEFINED.

Attributes

DC CIPAE is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME_GDI is implemented:

Together with the NSE2 and NSE field, this field specifies the target physical address space.

NSE2	NSE	NS	Meaning
0b0	0b0	0b0	Reserved.
0b0	0b0	0b1	Reserved.
0b0	0b1	0b0	Reserved.
0b0	0b1	0b1	Realm.
0b1	0b0	0b0	System Agent.
0b1	0b0	0b1	NS Protected.
0b1	0b1	0b0	Reserved.
0b1	0b1	0b1	Reserved.

If {NSE2, NSE, NS} is reserved, then no cache entries are required to be cleaned or invalidated.

Otherwise:

Together with the NSE field, this field specifies the target physical address space.

NSE	NS	Meaning
0b0	0b0	Reserved.
0b0	0b1	Reserved.
0b1	0b0	Reserved.
0b1	0b1	Realm.

If {NSE, NS} is not {1, 1}, then no cache entries are required to be cleaned or invalidated.

NSE, bit [62]

If FEAT_RME_GDI is implemented, this field together with the NS and NSE2 fields, specifies the target physical address space.

Otherwise, this field and the NS field specify the physical address space

For a description of the values derived by evaluating NS, NSE, and NSE2 together, see DC CIPAE.NS.

NSE2, bit [61]

When FEAT_RME_GDI is implemented:

Together with the NS and NSE field, this field specifies the target physical address space.

For a description of the values derived by evaluating NS and NSE together, see DC CIPAE.NS.

Otherwise:

Reserved, RES0.

Bits [60:56]

Reserved, RES0.

PA[55:52], bits [55:52]

When FEAT_D128 is implemented:

Extension to PA[51:0] if ID_AA64MMFR0_EL1.PARange = 0111. For more information see PA[51:0].

Otherwise:

Reserved, RES0.

PA, bits [51:0]

Physical address to use. No alignment restrictions apply to this PA.

Executing DC CIPAE

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIPAE, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b0111	0b1110	0b000

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoE);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoE);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIPAPA, Data or unified Cache line Clean and Invalidate by PA to PoPA

The DC CIPAPA characteristics are:

Purpose

Clean and Invalidate data cache by physical address to the Point of Physical Aliasing.

Note

This instruction cleans and invalidates all copies of the Location specified in the Xt argument, irrespective of any MECID associated with the Location. Memory accesses resulting from the Clean operation use the MECID associated with the cache entry.

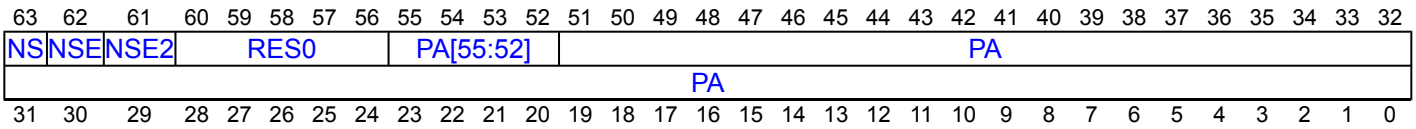
Configuration

This instruction is present only when FEAT_RME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CIPAPA are UNDEFINED.

Attributes

DC CIPAPA is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME_GDI is implemented:

Together with the NSE2 and NSE field, this field specifies the target physical address space.

NSE2	NSE	NS	Meaning
0b0	0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b0	0b1	Non-secure.
0b0	0b1	0b0	Root.
0b0	0b1	0b1	Realm.
0b1	0b0	0b0	System Agent.
0b1	0b0	0b1	NS Protected.
0b1	0b1	0b0	Reserved.
0b1	0b1	0b1	Reserved.

If {NSE2, NSE, NS} is reserved, then no cache entries are required to be cleaned or invalidated.

Otherwise:

Together with the NSE field, this field specifies the target physical address space.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

If FEAT_SEL2 is not implemented, and {NSE, NS} == {0b0, 0b0}, then no cache entries are required to be cleaned or invalidated

NSE, bit [62]

If FEAT_RME_GDI is implemented, this field together with the NS and NSE2 fields, specifies the target physical address space.

Otherwise, this field and the NS field specify the physical address space

For a description of the values derived by evaluating NS, NSE, and NSE2 together, see DC CIPAPA.NS.

NSE2, bit [61]

When FEAT_RME_GDI is implemented:

Together with the NS and NSE field, this field specifies the target physical address space.

For a description of the values derived by evaluating NS and NSE together, see DC CIPAPA.NS.

Otherwise:

Reserved, RES0.

Bits [60:56]

Reserved, RES0.

PA[55:52], bits [55:52]

When FEAT_D128 is implemented:

Extension to PA[51:0] if ID_AA64MMFR0_EL1.PARange = 0111. For more information see PA[51:0].

Otherwise:

Reserved, RES0.

PA, bits [51:0]

Physical address to use. No alignment restrictions apply to this PA.

Executing DC CIPAPA

- This instruction is not subject to any translation, permission checks, or granule protection checks.
- This instruction affects all caches in the Outer Shareable shareability domain.
- This instruction has the same ordering, observability, and completion behavior as VA-based cache maintenance instructions issued to the Outer Shareable shareability domain.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CIPAPA, <Xt>

op0	op1	CRn	CRm	op2
0b01	0b110	0b0111	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoPA);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CISW, Data or unified Cache line Clean and Invalidate by Set/Way

The DC CISW characteristics are:

Purpose

Clean and Invalidate data cache by set/way.

Configuration

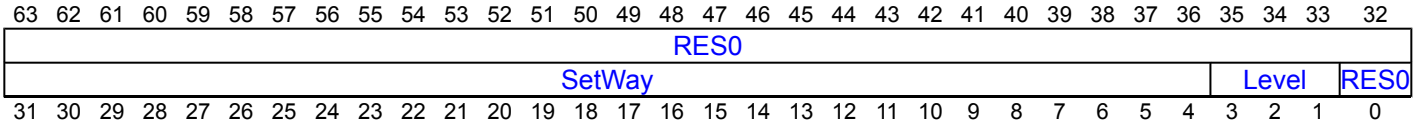
AArch64 System instruction DC CISW performs the same function as AArch32 System instruction DCCISW.

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DC CISW are UNDEFINED.

Attributes

DC CISW is a 64-bit System instruction.

Field descriptions



Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DC CISW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CISW, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1110	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIVAC, Data or unified Cache line Clean and Invalidate by VA to PoC

The DC CIVAC characteristics are:

Purpose

Clean and Invalidate data cache by address to Point of Coherency.

Configuration

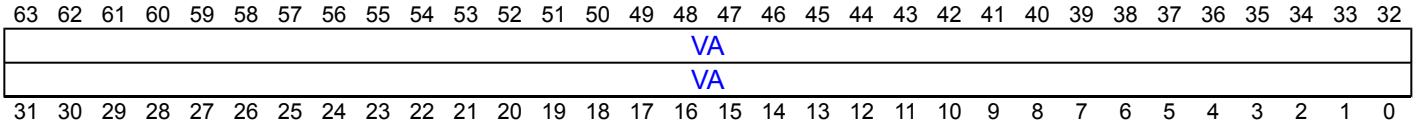
AArch64 System instruction DC CIVAC performs the same function as AArch32 System instruction DCCIMVAC.

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DC CIVAC are UNDEFINED.

Attributes

DC CIVAC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CIVAC

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIVAC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1110	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIVAOC, Data or unified Cache line Clean and Invalidate by VA to Outer Cache level

The DC CIVAOC characteristics are:

Purpose

Clean and Invalidate data cache by address to Outer cache level.

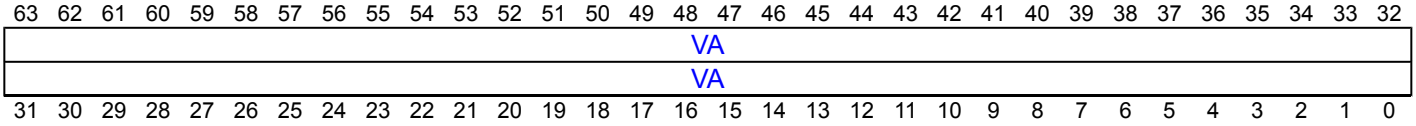
Configuration

This instruction is present only when FEAT_OCCMO is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CIVAOC are UNDEFINED.

Attributes

DC CIVAOC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CIVAOC

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIVAOC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1111	0b000

```
if !(IsFeatureImplemented(FEAT_OCCMO) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_OuterCache);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIVAPS, Clean and Invalidate of Data by VA to PoPS

The DC CIVAPS characteristics are:

Purpose

Clean and Invalidate data in data cache by virtual address to the Point of Physical Storage.

Note

This instruction cleans and invalidates all copies of the Location specified in the Xt argument, irrespective of any MECID associated with the Location. Memory accesses resulting from the Clean operation use the MECID associated with the cache entry.

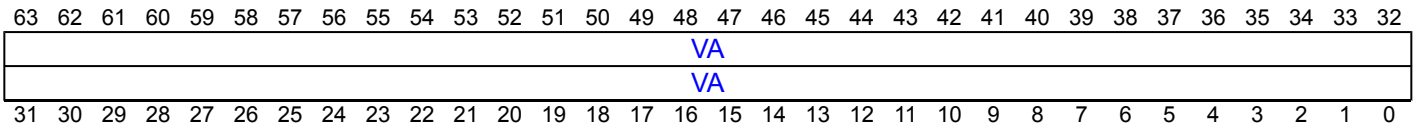
Configuration

This instruction is present only when FEAT_PoPS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CIVAPS are UNDEFINED.

Attributes

DC CIVAPS is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CIVAPS

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CIVAPS, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1111	0b001

```
if !(IsFeatureImplemented(FEAT_PoPS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGITR2_EL2.nDCCIVAPS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoPS);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoPS);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoPS);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CSW, Data or unified Cache line Clean by Set/Way

The DC CSW characteristics are:

Purpose

Clean data cache by set/way.

Configuration

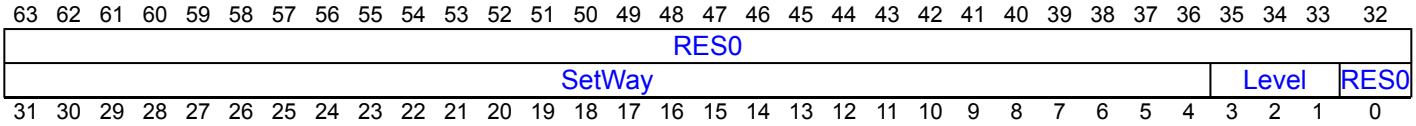
AArch64 System instruction DC CSW performs the same function as AArch32 System instruction DCCSW.

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DC CSW are UNDEFINED.

Attributes

DC CSW is a 64-bit System instruction.

Field descriptions



Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DC CSW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
    - No cache lines.
    - A single arbitrary cache line.
    - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CSW, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b1010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CVAC, Data or unified Cache line Clean by VA to PoC

The DC CVAC characteristics are:

Purpose

Clean data cache by address to Point of Coherency.

Configuration

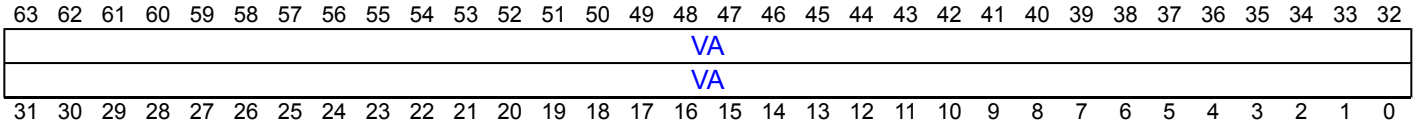
AArch64 System instruction DC CVAC performs the same function as AArch32 System instruction DCCMVAC.

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DC CVAC are UNDEFINED.

Attributes

DC CVAC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CVAC

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CVAC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CVADP, Data or unified Cache line Clean by VA to PoDP

The DC CVADP characteristics are:

Purpose

Clean data cache by address to Point of Deep Persistence.

If the memory system does not identify a Point of Deep Persistence, then this instruction behaves as a DC CVAP.

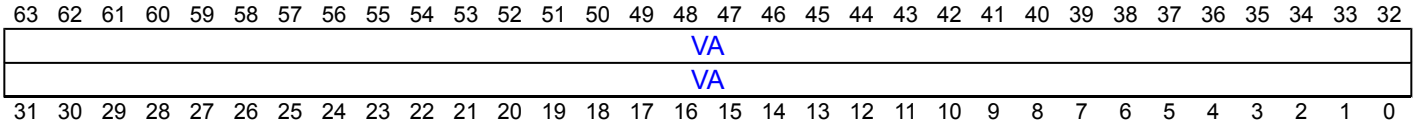
Configuration

This instruction is present only when FEAT_DPB2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CVADP are UNDEFINED.

Attributes

DC CVADP is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CVADP

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CVADP, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1101	0b001

```
if !(IsFeatureImplemented(FEAT_DPB2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoDP);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVADP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoDP);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoDP);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoDP);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CVAOC, Data or unified Cache line Clean by VA to Outer Cache level

The DC CVAOC characteristics are:

Purpose

Clean data cache by address to Outer cache level.

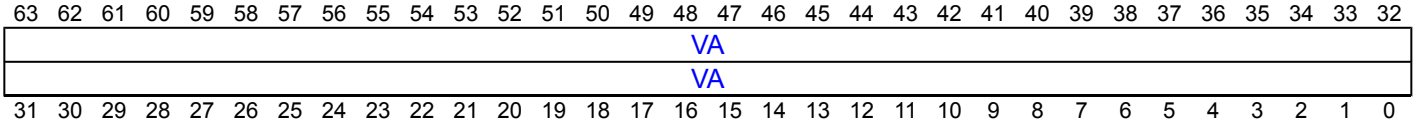
Configuration

This instruction is present only when FEAT_OCCMO is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CVAOC are UNDEFINED.

Attributes

DC CVAOC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CVAOC

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CVAOC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1011	0b000

```
if !(IsFeatureImplemented(FEAT_OCCMO) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_OuterCache);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_OuterCache);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CVAP, Data or unified Cache line Clean by VA to PoP

The DC CVAP characteristics are:

Purpose

Clean data cache by address to Point of Persistence.

If the memory system does not identify a Point of Persistence, then this instruction behaves as a DC CVAC.

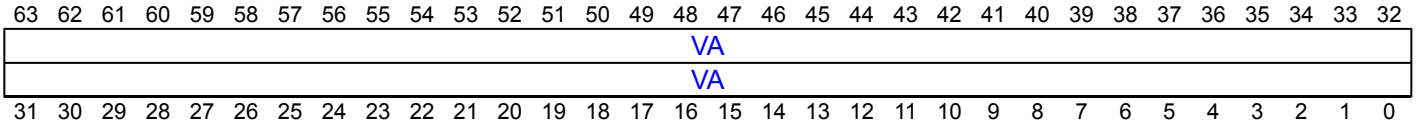
Configuration

This instruction is present only when FEAT_DPB is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DC CVAP are UNDEFINED.

Attributes

DC CVAP is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CVAP

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CVAP, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_DPB) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoP);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVAP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoP);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoP);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoP);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CVAU, Data or unified Cache line Clean by VA to PoU

The DC CVAU characteristics are:

Purpose

Clean data cache by address to Point of Unification.

Configuration

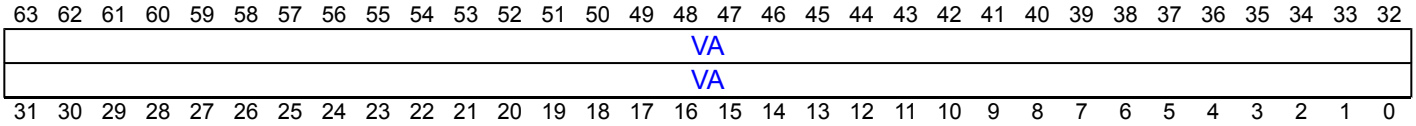
AArch64 System instruction DC CVAU performs the same function as AArch32 System instruction DCCMVAU.

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DC CVAU are UNDEFINED.

Attributes

DC CVAU is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC CVAU

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC CVAU, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b1011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TOCU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoU);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TOCU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCCVAU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoU);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoU);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Clean, CacheOpScope_PoU);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC GVA, Data Cache set Allocation Tag by VA

The DC GVA characteristics are:

Purpose

Write a value to the Allocation Tags of a naturally aligned block of N bytes, where the size of N is identified in DCZID_EL0. The Allocation Tag used is determined by the input address.

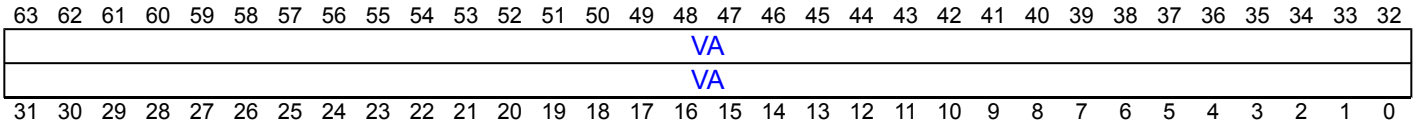
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC GVA are UNDEFINED.

Attributes

DC GVA is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. There is no alignment restriction on the address within the block of N bytes that is used.

Executing DC GVA

When this instruction is executed, it can generate memory faults or watchpoints which are prioritized in the same way as other memory-related faults or watchpoints. If a synchronous Data Abort fault or a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is not set.

A DC GVA instruction to any type of Device memory is CONSTRAINED UNPREDICTABLE between:

- Not modifying Allocation tags at the specified addresses.
- Generating an Alignment Fault determined by the memory type.

This instruction applies to Normal memory regardless of cacheability attributes.

This instruction behaves as a set of stores to each Allocation Tag within the block being accessed, and so it:

- Generates a Permission fault if the translation system does not permit writes to the locations.
- Requires the same considerations for ordering and the management of coherency as any other store instructions.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC GVA, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0100	0b011

```
if !IsFeatureImplemented(FEAT_MTE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.DZE == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TDZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCZVA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.DZE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.MemZero(X[t, 64], CacheType_Tag);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TDZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCZVA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.MemZero(X[t, 64], CacheType_Tag);
elsif PSTATE.EL == EL2 then
    AArch64.MemZero(X[t, 64], CacheType_Tag);
elsif PSTATE.EL == EL3 then
    AArch64.MemZero(X[t, 64], CacheType_Tag);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC GZVA, Data Cache set Allocation Tags and Zero by VA

The DC GZVA characteristics are:

Purpose

Zero data and write a value to the Allocation Tags of a naturally aligned block of N bytes, where the size of N is identified in DCZID_EL0. The Allocation Tag used is determined by the input address.

This instruction might have the poison-atomic property for IMPLEMENTATION DEFINED regions of physical memory.

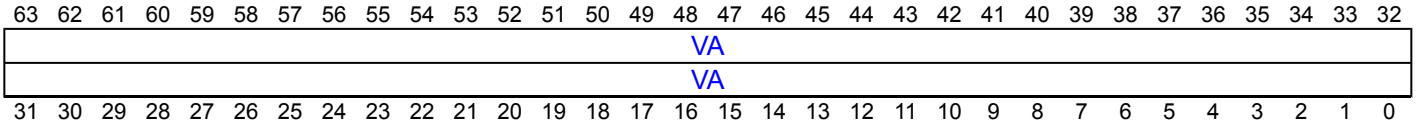
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC GZVA are UNDEFINED.

Attributes

DC GZVA is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. There is no alignment restriction on the address within the block of N bytes that is used.

Executing DC GZVA

When this instruction is executed, it can generate memory faults or watchpoints which are prioritized in the same way as other memory-related faults or watchpoints. If a synchronous Data Abort fault or a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is not set.

If the memory region being zeroed is any type of Device memory, this instruction generates an alignment fault which is prioritized in the same way as other alignment faults that are determined by the memory type.

This instruction applies to Normal memory regardless of cacheability attributes.

This instruction behaves as a set of Stores to each byte and Allocation tag within the block being accessed, and so it:

- Generates a Permission fault if the translation system does not permit writes to the locations.
- Requires the same considerations for ordering and the management of coherency as any other store instructions.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC GZVA, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_MTE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.DZE == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TDZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCZVA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.DZE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.MemZero(X[t, 64], CacheType_Data_Tag);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TDZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCZVA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.MemZero(X[t, 64], CacheType_Data_Tag);
elsif PSTATE.EL == EL2 then
    AArch64.MemZero(X[t, 64], CacheType_Data_Tag);
elsif PSTATE.EL == EL3 then
    AArch64.MemZero(X[t, 64], CacheType_Data_Tag);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC IGDSW, Invalidate of Data and Allocation Tags by Set/Way

The DC IGDSW characteristics are:

Purpose

Invalidate data and Allocation Tags in data cache by set/way.

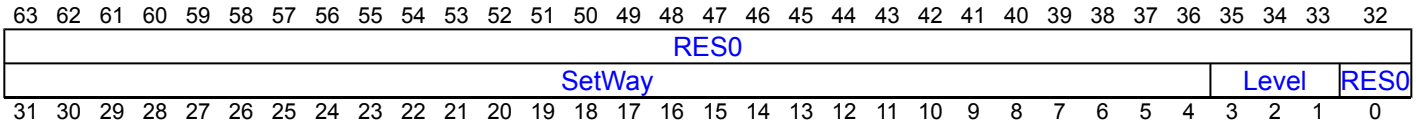
Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC IGDSW are UNDEFINED.

Attributes

DC IGDSW is a 64-bit System instruction.

Field descriptions



Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DC IGDSW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:

- No cache lines.
- A single arbitrary cache line.
- Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC IGDSW, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0110	0b110

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC IGDVAC, Invalidate of Data and Allocation Tags by VA to PoC

The DC IGDVAC characteristics are:

Purpose

Invalidate data and Allocation Tags in data cache by address to Point of Coherency.

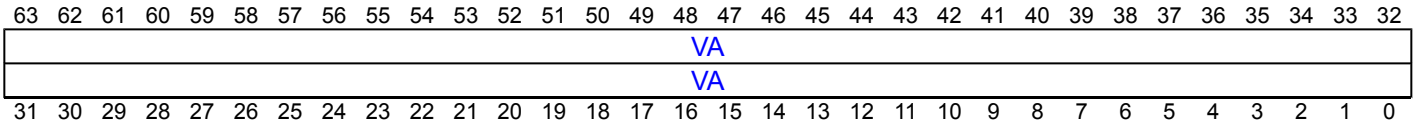
Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC IGDVAC are UNDEFINED.

Attributes

DC IGDVAC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC IGDVAC

When the instruction is executed, it can generate a watchpoint, which is prioritized in the same way as other watchpoints. If a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 1.

This instruction requires write access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC IGDVAC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0110	0b101

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC IGSW, Invalidate of Allocation Tags by Set/Way

The DC IGSW characteristics are:

Purpose

Invalidate Allocation Tags in data cache by set/way.

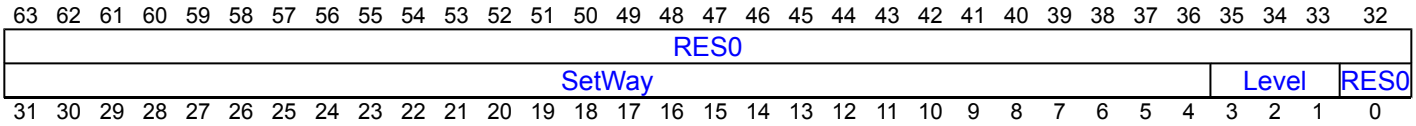
Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC IGSW are UNDEFINED.

Attributes

DC IGSW is a 64-bit System instruction.

Field descriptions



Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DC IGSW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:

- No cache lines.
- A single arbitrary cache line.
- Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC IGSW, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0110	0b100

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC IGVAC, Invalidate of Allocation Tags by VA to PoC

The DC IGVAC characteristics are:

Purpose

Invalidate Allocation Tags in data cache by address to Point of Coherency.

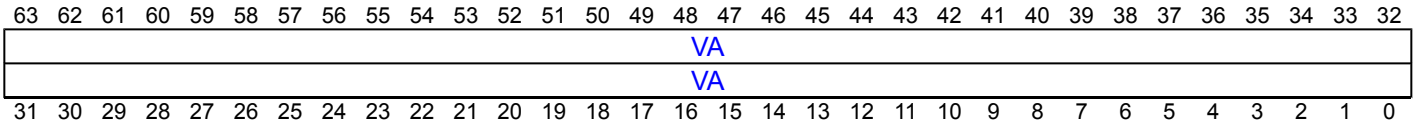
Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC IGVAC are UNDEFINED.

Attributes

DC IGVAC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC IGVAC

When the instruction is executed, it can generate a watchpoint, which is prioritized in the same way as other watchpoints. If a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 1.

This instruction requires write access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC IGVAC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0110	0b011

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC ISW, Data or unified Cache line Invalidate by Set/Way

The DC ISW characteristics are:

Purpose

Invalidate data cache by set/way.

When FEAT_MTE2 is implemented, this instruction might invalidate Allocation Tags from caches. When it invalidates Allocation Tags from caches, it also cleans them.

Configuration

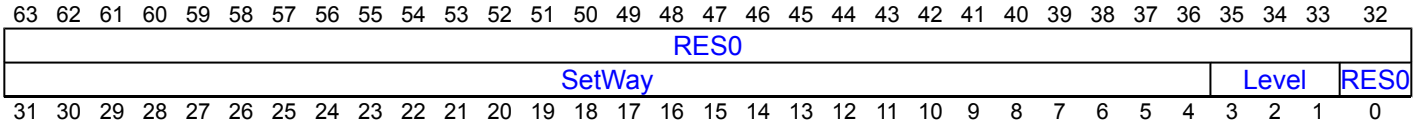
AArch64 System instruction DC ISW performs the same function as AArch32 System instruction DCISW.

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DC ISW are UNDEFINED.

Attributes

DC ISW is a 64-bit System instruction.

Field descriptions



Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DC ISW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
    - No cache lines.
    - A single arbitrary cache line.
    - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC ISW, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0110	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Invalidate, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC IVAC, Data or unified Cache line Invalidate by VA to PoC

The DC IVAC characteristics are:

Purpose

Invalidate data cache by address to Point of Coherency.

When FEAT_MTE2 is implemented, this instruction might invalidate Allocation Tags from caches. When it invalidates Allocation Tags from caches, it also cleans them.

Configuration

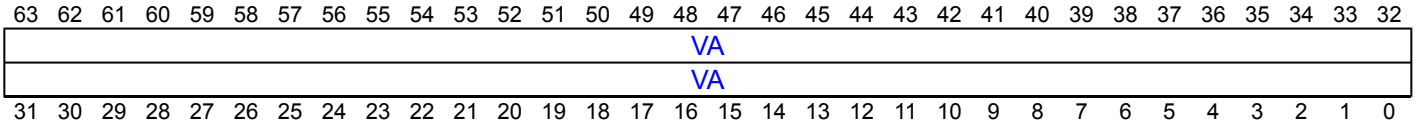
AArch64 System instruction DC IVAC performs the same function as AArch32 System instruction DCIMVAC.

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DC IVAC are UNDEFINED.

Attributes

DC IVAC is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DC IVAC

When the instruction is executed, it can generate a watchpoint, which is prioritized in the same way as other watchpoints. If a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 1.

This instruction requires write access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC IVAC, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0110	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t, 64], CacheType_Data, CacheOp_Invalidate, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC ZVA, Data Cache Zero by VA

The DC ZVA characteristics are:

Purpose

Zero data cache by address. Zeroes a naturally aligned block of N bytes, where the size of N is identified in DCZID_EL0.

This instruction might have the poison-atomic property for IMPLEMENTATION DEFINED regions of physical memory.

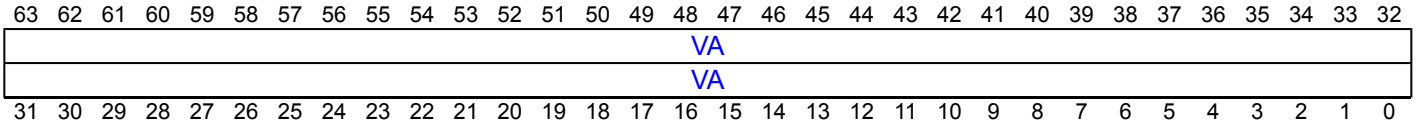
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DC ZVA are UNDEFINED.

Attributes

DC ZVA is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. There is no alignment restriction on the address within the block of N bytes that is used.

Executing DC ZVA

When this instruction is executed, it can generate memory faults or watchpoints which are prioritized in the same way as other memory-related faults or watchpoints. If a synchronous Data Abort fault or a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 0.

If the memory region being zeroed is any type of Device memory, this instruction generates an Alignment fault which is prioritized in the same way as other Alignment faults that are determined by the memory type.

This instruction applies to Normal memory regardless of cacheability attributes.

This instruction behaves as a set of Stores to each byte within the block being accessed, and so it:

- Generates a Permission fault if the translation system does not permit writes to the locations.
- Requires the same considerations for ordering and the management of coherency as any other store instructions.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DC ZVA, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.DZE == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TDZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCZVA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.DZE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.MemZero(X[t, 64], CacheType_Data);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TDZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DCZVA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.MemZero(X[t, 64], CacheType_Data);
elsif PSTATE.EL == EL2 then
    AArch64.MemZero(X[t, 64], CacheType_Data);
elsif PSTATE.EL == EL3 then
    AArch64.MemZero(X[t, 64], CacheType_Data);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCZID_EL0, Data Cache Zero ID Register

The DCZID_EL0 characteristics are:

Purpose

Indicates the block size that is written with byte values of 0 by the DC ZVA (Data Cache Zero by Address) System instruction.

If FEAT_MTE is implemented, this register also indicates the granularity at which the DC GVA and DC GZVA instructions write.

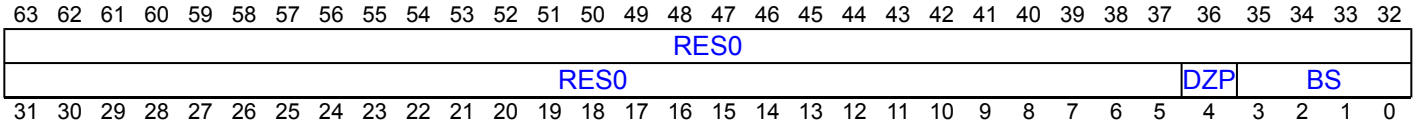
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DCZID_EL0 are UNDEFINED.

Attributes

DCZID_EL0 is a 64-bit register.

Field descriptions



Bits [63:5]

Reserved, RES0.

DZP, bit [4]

Data Zero Prohibited. This field indicates whether use of DC ZVA instructions is permitted or prohibited.

If FEAT_MTE is implemented, this field also indicates whether use of the DC GVA and DC GZVA instructions are permitted or prohibited.

DZP	Meaning
0b0	Instructions are permitted.
0b1	Instructions are prohibited.

The value read from this field is governed by the current Exception level and the values of the following fields:

- The Effective value of HCR_EL2.TDZ.
- When the Effective value of HCR_EL2.{E2H, TGE} != '11', SCTLR_EL1.DZE.
- When the Effective value of HCR_EL2.{E2H, TGE} == '11', SCTLR_EL2.DZE.

BS, bits [3:0]

Log_2 of the block size in words. The maximum size supported is 2KB, indicated by value 0b1001.

If FEAT_MTE2 is implemented, the minimum size supported is 16 bytes, indicated by value 0b0010.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing DCZID_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DCZID_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0000	0b0000	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGRTR_EL2.DCZID_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = DCZID_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.DCZID_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = DCZID_EL0;
elsif PSTATE.EL == EL2 then
    X[t, 64] = DCZID_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DCZID_EL0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DISR_EL1, Deferred Interrupt Status Register

The DISR_EL1 characteristics are:

Purpose

Records that an SError exception has been consumed by an ESB instruction.

Configuration

AArch64 System register DISR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DISR[31:0].

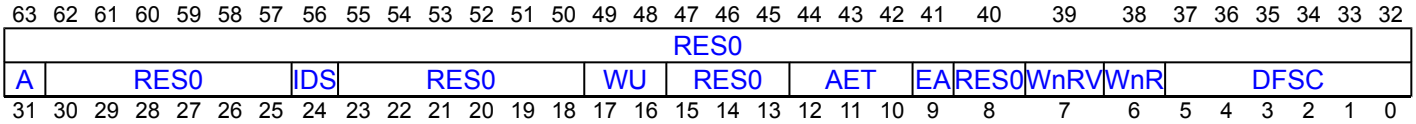
This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to DISR_EL1 are UNDEFINED.

Attributes

DISR_EL1 is a 64-bit register.

Field descriptions

When DISR_EL1.IDS == 0:



Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError exception. If the implementation does not include any sources of SError exception that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

Indicates the deferred SError exception type.

IDS	Meaning
0b0	Deferred error uses architecturally-defined format.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:18]

Reserved, RES0.

WU, bits [17:16]

When FEAT_RASv2 is implemented:

Write update. See the description of ESR_ELx.WU for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:13]

Reserved, RES0.

AET, bits [12:10]

Asynchronous Error Type. See the description of ESR_ELx.AET for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort Type. See the description of ESR_ELx.EA for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

WnRV, bit [7]

When FEAT_RASv2 is implemented:

Write-not-Read Valid. See the description of ESR_ELx.WnRV for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WnR, bit [6]

When FEAT_RASv2 is implemented:

Write-not-Read. See the description of ESR_ELx.WnR for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

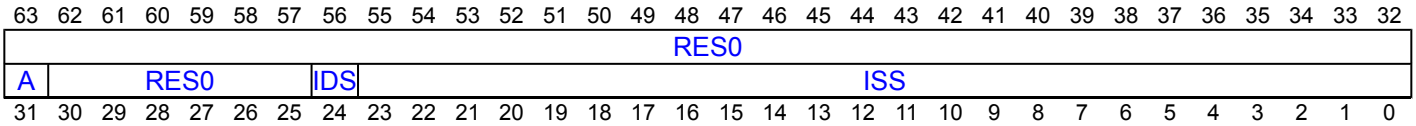
DFSC, bits [5:0]

Fault Status Code. See the description of ESR_ELx.DFSC for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When DISR_EL1.IDS == 1:



Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError exception. If the implementation does not include any sources of SError exception that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

Indicates the deferred SError exception type.

IDS	Meaning
0b1	Deferred error uses IMPLEMENTATION DEFINED format.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

IMPLEMENTATION DEFINED syndrome. See the description of ESR_ELx[23:0] for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DISR_EL1

An indirect write to DISR_EL1 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR_EL1 occurring in program order after the ESB instruction.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DISR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        X[t, 64] = VDISR_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DISR_EL1;
```

MSR DISR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        VDISR_EL2 = X[t, 64];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DIT, Data Independent Timing

The DIT characteristics are:

Purpose

Allows access to the Data Independent Timing bit.

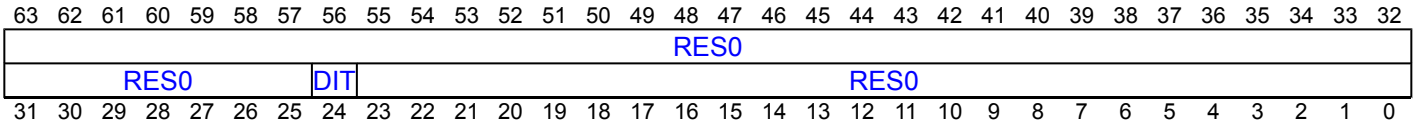
Configuration

This register is present only when FEAT_DIT is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DIT are UNDEFINED.

Attributes

DIT is a 64-bit register.

Field descriptions



Bits [63:25]

Reserved, RES0.

DIT, bit [24]

Data Independent Timing.

DIT	Meaning
0b0	The architecture makes no statement about the timing properties of any instructions.
0b1	The architecture requires that: • The timing of every load and store instruction is insensitive to the value of the data being loaded or stored. • For certain data processing instructions, the instruction takes a time which is independent of: ◦ The values of the data supplied in any of its registers. ◦ The values of the NZCV flags. • For certain data processing instructions, the response of the instruction to asynchronous exceptions does not vary based on: ◦ The values of the data supplied in any of its registers. ◦ The values of the NZCV flags.

The instructions affected by this bit are:

- Branches, Exception Generating and System instructions: CFINV, and NOP.
- Data Processing -- Immediate:
  - Add/subtract (immediate): ADD, ADDS, SUB, and SUBS.
  - Bitfield: BFM, SBFM, and UBFM.
  - Extract: EXTR.
  - Logical (immediate): AND, ANDS, EOR, and ORR.
  - Min/max (immediate): SMAX, SMIN, UMAX, and UMIN.
  - Move wide (immediate): MOVK, MOVN, and MOVZ.
- Data Processing -- Register:
  - Add/subtract (extended register): ADD, ADDS, SUB, and SUBS.

    - Add/subtract (shifted register): `ADD`, `ADDS`, `SUB`, and `SUBS`.
    - Add/subtract (with carry): `ADC`, `ADCS`, `SBC`, and `SBCS`.
    - Conditional compare (immediate): `CCMN`, and `CCMP`.
    - Conditional compare (register): `CCMN`, and `CCMP`.
    - Conditional select: `CSEL`, `CSINC`, `CSINV`, and `CSNEG`.
    - Data-processing (1 source): `ABS`, `CLS`, `CLZ`, `CNT`, `CTZ`, `RBIT`, `REV16`, `REV32`, and `REV`.
    - Data-processing (2 source): `ASRV`, `CRC32B`, `CRC32CB`, `CRC32CH`, `CRC32CW`, `CRC32CX`, `CRC32H`, `CRC32W`, `CRC32X`, `LSLV`, `LSRV`, `RORV`, `SMAX`, `SMIN`, `UMAX`, and `UMIN`.
    - Data-processing (3 source): `MADD`, `MSUB`, `SMADDL`, `SMSUBL`, `SMULH`, `UMADDL`, `UMSUBL`, and `UMULH`.
    - Evaluate into flags: `SETF16`, and `SETF8`.
    - Logical (shifted register): `AND`, `ANDS`, `BIC`, `BICS`, `EON`, `EOR`, `ORN`, and `ORR`.
    - Rotate right into flags: `RMIF`.
- Data Processing -- Scalar Floating-Point and Advanced SIMD:
    - Advanced SIMD across lanes: `ADDV`, `SADDLV`, `SMAXV`, `SMINV`, `UADDLV`, `UMAXV`, and `UMINV`.
    - Advanced SIMD copy: `DUP`, `INS`, `SMOV`, and `UMOV`.
    - Advanced SIMD extract: `EXT`.
    - Advanced SIMD modified immediate: `BIC`, `MOVI`, `MVNI`, and `ORR`.
    - Advanced SIMD permute: `TRN1`, `TRN2`, `UZP1`, `UZP2`, `ZIP1`, and `ZIP2`.
    - Advanced SIMD scalar copy: `DUP`.
    - Advanced SIMD scalar pairwise: `ADDP`.
    - Advanced SIMD scalar shift by immediate: Advanced SIMD scalar shift by immediate: `SHL`, `SLI`, `SRSHR`, `SRI`, `SRSRA`, `SSHR`, `SSRA`, `URSHR`, `URSRA`, `USHR`, and `USRA`.
    - Advanced SIMD scalar three same: `ADD`, `CMEQ`, `CMGE`, `CMGT`, `CMHI`, `CMHS`, `CMTST`, `SQDMULH`, `SQRDMULH`, `SSHL`, `SUB`, and `USHL`.
    - Advanced SIMD scalar three same extra: `SQRDMLAH`.
    - Advanced SIMD scalar two-register miscellaneous: `ABS`, `CMEQ`, `CMGE`, `CMGT`, `CMLE`, `CMLT`, and `NEG`.
    - Advanced SIMD scalar x indexed element: `SQDMULH`, `SQRDMLAH`, and `SQRDMULH`.
    - Advanced SIMD shift by immediate: `RSHRN2`, `RSHRN`, `SHL`, `SHRN2`, `SHRN`, `SLI`, `SRI`, `SSHLL2`, `SSHLL`, `SSHR`, `SSRA`, `USHLL2`, `USHLL`, `USHR`, and `USRA`.
    - Advanced SIMD table lookup: `LUTI2`, `LUTI4`, `TBL`, and `TBX`.
    - Advanced SIMD three different: `ADDHN2`, `ADDHN`, `PMULL2`, `PMULL`, `RADDHN2`, `RADDHN`, `RSUBHN2`, `RSUBHN`, `SABAL2`, `SABAL`, `SABDL2`, `SABDL`, `SADDL2`, `SADDL`, `SADDW2`, `SADDW`, `SMLAL2`, `SMLAL`, `SMLSL2`, `SMLSL`, `SMULL2`, `SMULL`, `SSUBL2`, `SSUBL`, `SSUBW2`, `SSUBW`, `SUBHN2`, `SUBHN`, `UABAL2`, `UABAL`, `UABDL2`, `UABDL`, `UADDL2`, `UADDL`, `UADDW2`, `UADDW`, `UMLAL2`, `UMLAL`, `UMLSL2`, `UMLSL`, `UMULL2`, `UMULL`, `USUBL2`, `USUBL`, `USUBW2`, and `USUBW`.
    - Advanced SIMD three same: `ADD`, `ADDP`, `AND`, `BIC`, `BIF`, `BIT`, `BSL`, `CMEQ`, `CMGE`, `CMGT`, `CMHI`, `CMHS`, `CMTST`, `EOR`, `MLA`, `MLS`, `MUL`, `ORN`, `ORR`, `PMUL`, `SABA`, `SABD`, `SHADD`, `SHSUB`, `SMAX`, `SMAXP`, `SMIN`, `SMINP`, `SQDMULH`, `SQRDMULH`, `SSHL`, `SUB`, `UABA`, `UABD`, `UHADD`, `UHSUB`, `UMAX`, `UMAXP`, `UMIN`, `UMINP`, and `USHL`.
    - Advanced SIMD three-register extension: `SDOT`, `SMMLA`, `SQRDMLAH`, `UDOT`, `UMMLA`, `USDOT`, and `USMMLA`.
    - Advanced SIMD two-register miscellaneous: `ABS`, `CLS`, `CLZ`, `CMEQ`, `CMGE`, `CMGT`, `CMLE`, `CMLT`, `CNT`, `NEG`, `NOT`, `RBIT`, `REV16`, `REV32`, `REV64`, `SADALP`, `SADDLP`, `SHLL2`, `SHLL`, `UADALP`, `UADDLP`, `XTN2`, and `XTN`.
    - Advanced SIMD vector x indexed element: `MLA`, `MLS`, `MUL`, `SDOT`, `SMLAL2`, `SMLAL`, `SMLSL2`, `SMLSL`, `SMULL2`, `SMULL`, `SQDMULH`, `SQRDMLAH`, `SQRDMULH`, `SUDOT`, `UDOT`, `UMLAL2`, `UMLAL`, `UMLSL2`, `UMLSL`, `UMULL2`, `UMULL`, and `USDOT`.
    - Cryptographic AES: `AESD`, `AESE`, `AESIMC`, and `AESMC`.
    - Cryptographic four-register: `BCAX`, `EOR3`, and `SM3SS1`.
    - Cryptographic three-register SHA: `SHA1C`, `SHA1M`, `SHA1P`, `SHA1SU0`, `SHA256H2`, `SHA256H`, and `SHA256SU1`.
    - Cryptographic three-register SHA 512: `RAX1`, `SHA512H2`, `SHA512H`, `SHA512SU1`, `SM3PARTW1`, `SM3PARTW2`, and `SM4EKEY`.
    - Cryptographic three-register, imm2: `SM3TT1A`, `SM3TT1B`, `SM3TT2A`, and `SM3TT2B`.
    - Cryptographic three-register, imm6: `XAR`.
    - Cryptographic two-register SHA: `SHA1H`, `SHA1SU1`, and `SHA256SU0`.
    - Cryptographic two-register SHA 512: `SHA512SU0`, and `SM4E`.
    - Floating-point conditional select: `FCSEL`.
- Loads and Stores:
    - 128-bit atomic memory operations: `LDCLRP`, `LDCLRPA`, `LDCLRPAL`, `LDCLRPL`, `LDSETP`, `LDSETPA`, `LDSETPAL`, `LDSETPL`, `SWPP`, `SWPPA`, `SWPPAL`, and `SWPPL`.
    - Advanced SIMD load/store multiple structures: `LD1`, `LD2`, `LD3`, `LD4`, `ST1`, `ST2`, `ST3`, and `ST4`.
    - Advanced SIMD load/store multiple structures (post-indexed): `LD1`, `LD2`, `LD3`, `LD4`, `ST1`, `ST2`, `ST3`, and `ST4`.
    - Advanced SIMD load/store single structure: `LD1`, `LD1R`, `LD2`, `LD2R`, `LD3`, `LD3R`, `LD4`, `LD4R`, `LDAP1`, `ST1`, `ST2`, `ST3`, `ST4`, and `STL1`.
    - Advanced SIMD load/store single structure (post-indexed): `LD1`, `LD1R`, `LD2`, `LD2R`, `LD3`, `LD3R`, `LD4`, `LD4R`, `ST1`, `ST2`, `ST3`, and `ST4`.
    - Atomic memory operations: `LDADD`, `LDADDA`, `LDADDAB`, `LDADDAH`, `LDADDAL`, `LDADDALB`, `LDADDALH`, `LDADDB`, `LDADDH`, `LDADDL`, `LDADDLB`, `LDADDLH`, `LDAPR`, `LDAPRB`, `LDAPRH`, `LDCLR`, `LDCLRA`, `LDCLRAB`, `LDCLRAH`, `LDCLRAL`, `LDCLRALB`, `LDCLRALH`, `LDCLRB`, `LDCLRH`, `LDCLRL`, `LDCLRLB`, `LDCLRLH`, `LDEOR`, `LDEORA`, `LDEORAB`, `LDEORAH`, `LDEORAL`, `LDEORALB`, `LDEORALH`, `LDEORB`, `LDEORH`, `LDEORL`, `LDEORLB`, `LDEORLH`, `LDSET`, `LDSETA`,

LDSETAB, LDSETAH, LDSETAL, LDSETALB, LDSETALH, LDSETB, LDSETH, LDSETL, LDSETLB, LDSETLH, LDSMAX, LDSMAXA, LDSMAXAB, LDSMAXAH, LDSMAXAL, LDSMAXALB, LDSMAXALH, LDSMAXB, LDSMAXH, LDSMAXL, LDSMAXLB, LDSMAXLH, LDSMIN, LDSMINA, LDSMINAB, LDSMINAH, LDSMINAL, LDSMINALB, LDSMINALH, LDSMINB, LDSMINH, LDSMINL, LDSMINLB, LDSMINLH, LDUMAX, LDUMAXA, LDUMAXAB, LDUMAXAH, LDUMAXAL, LDUMAXALB, LDUMAXALH, LDUMAXB, LDUMAXH, LDUMAXL, LDUMAXLB, LDUMAXLH, LDUMIN, LDUMINA, LDUMINAB, LDUMINAH, LDUMINAL, LDUMINALB, LDUMINALH, LDUMINB, LDUMINH, LDUMINL, LDUMINLB, and LDUMINLH.
    - LDAPR/STLR (SIMD&FP): LDAPUR, and STLUR.
    - LDAPR/STLR (unscaled immediate): LDAPUR, LDAPURB, LDAPURH, LDAPURSB, LDAPURSH, LDAPURSW, STLUR, STLURB, and STLURH.
    - LDAPR/STLR (writeback): LDAPR, and STLR.
    - LDIAPP/STILP: LDIAPP, and STILP.
    - Load register (literal): LDR, and LDRSW.
    - Load/store exclusive pair: LDAXP, LDXP, STLXP, and STXP.
    - Load/store exclusive register: LDAXR, LDAXRB, LDAXRH, LDXR, LDXRB, LDXRH, STLXR, STLXRB, STLXRH, STXR, STXRB and STXRH.
    - Load/store no-allocate pair (offset): LDNP, and STNP.
    - Load/store ordered: LDAR, LDARB, LDARH, LDLAR, LDLARB, LDLARH, STLLR, STLLRB, STLLRH, STLR, STLRB, and STLRH.
    - Load/store register (immediate post-indexed): LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, STR, STRB, and STRH.
    - Load/store register (immediate pre-indexed): LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, STR, STRB, and STRH.
    - Load/store register (pac): LDRAA, and LDRAB.
    - Load/store register (register offset): LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, STR, STRB, and STRH.
    - Load/store register (unprivileged): LDTR, LDTRB, LDTRH, LDTRSB, LDTRSH, LDTRSW, STTR, STTRB, and STTRH.
    - Load/store register (unscaled immediate): LDUR, LDURB, LDURH, LDURSB, LDURSH, LDURSW, STUR, STURB, and STURH.
    - Load/store register (unsigned immediate): LDR, LDRB, LDRH, LDRSB, LDRSH, LDRSW, STR, STRB, and STRH.
    - Load/store register pair (offset): LDP, LDPSW, and STP.
    - Load/store register pair (post-indexed): LDP, LDPSW, and STP.
    - Load/store register pair (pre-indexed): LDP, LDPSW, and STP.
- SME encodings:
    - SME Add Vector to Array: ADDHA, and ADDVA.
    - SME Integer Outer Product - 32 bit: SMOPA, SMOPS, SUMOPA, SUMOPS, UMOPA, UMOPS, USMOPA, and USMOPS.
    - SME Memory: LD1B, LD1D, LD1H, LD1Q, LD1W, LDR, ST1B, ST1D, ST1H, ST1Q, ST1W, and STR.
    - SME Move from Array: MOVA, and MOVAZ.
    - SME Move into Array: MOVA.
    - SME Outer Product - 64 bit:
        - SME Int16 outer product: SMOPA, SMOPS, SUMOPA, SUMOPS, UMOPA, UMOPS, USMOPA, and USMOPS.
    - SME zero array: ZERO.
    - SME2 Expand Lookup Table (Contiguous): LUTI2, and LUTI4.
    - SME2 Expand Lookup Table (Non-contiguous): LUTI2, and LUTI4.
    - SME2 Move Lookup Table: MOVT.
    - SME2 Multi-vector - Indexed (Four registers):
        - SME2 multi-vec indexed long MLA four sources: SMLAL, SMLSL, UMLAL, and UMLSL.
        - SME2 multi-vec indexed long long MLA four sources 32-bit: SMLALL, SMLSLL, SUMLALL, UMLALL, UMLSLL, and USMLALL.
        - SME2 multi-vec indexed long long MLA four sources 64-bit: SMLALL, SMLSLL, UMLALL, and UMLSLL.
        - SME2 multi-vec ternary indexed four registers 32-bit: SDOT, SUDOT, SUVDOT, SVDOT, UDOT, USDOT, USVDOT, and UVDOT.
        - SME2 multi-vec ternary indexed four registers 64-bit: SDOT, SVDOT, UDOT, and UVDOT.
    - SME2 Multi-vector - Indexed (One register):
        - SME2 multi-vec indexed long MLA one source: SMLAL, SMLSL, UMLAL, and UMLSL.
        - SME2 multi-vec indexed long long MLA one source 32-bit: SMLALL, SMLSLL, SUMLALL, UMLALL, UMLSLL, and USMLALL.
        - SME2 multi-vec indexed long long MLA one source 64-bit: SMLALL, SMLSLL, UMLALL, and UMLSLL.
    - SME2 Multi-vector - Indexed (Two registers):
        - SME2 multi-vec indexed long MLA two sources: SMLAL, SMLSL, UMLAL, and UMLSL.
        - SME2 multi-vec indexed long long MLA two sources 32-bit: SMLALL, SMLSLL, SUMLALL, UMLALL, UMLSLL, and USMLALL.
        - SME2 multi-vec indexed long long MLA two sources 64-bit: SMLALL, SMLSLL, UMLALL, and UMLSLL.
        - SME2 multi-vec ternary indexed two registers 32-bit: SDOT, SUDOT, SVDOT, UDOT, USDOT, and UVDOT.
        - SME2 multi-vec ternary indexed two registers 64-bit: SDOT, and UDOT.
    - SME2 Multi-vector - Memory (Contiguous): LD1B, LD1D, LD1H, LD1W, LDNT1B, LDNT1D, LDNT1H, LDNT1W, ST1B, ST1D, ST1H, ST1W, STNT1B, STNT1D, STNT1H, and STNT1W.
    - SME2 Multi-vector - Memory (Strided): LD1B, LD1D, LD1H, LD1W, LDNT1B, LDNT1D, LDNT1H, LDNT1W, ST1B, ST1D, ST1H, ST1W, STNT1B, STNT1D, STNT1H, and STNT1W.

    - SME2 Multi-vector - Multiple Array Vectors (Four registers):
        - SME2 multiple vectors binary int four registers: `ADD`, and `SUB`.
        - SME2 multiple vectors four-way dot product four registers: `SDOT`, and `UDOT`.
        - SME2 multiple vectors long MLA four sources: `SMLAL`, `SMLSL`, `UMLAL`, and `UMLSL`.
        - SME2 multiple vectors long long MLA four sources: `SMLALL`, `SMLSLL`, `UMLALL`, `UMLSLL`, and `USMLALL`.
        - SME2 multiple vectors mixed dot product four registers: `USDOT`.
        - SME2 multiple vectors ternary int four registers: `ADD`, and `SUB`.
        - SME2 multiple vectors two-way dot product four registers: `SDOT`, and `UDOT`.
    - SME2 Multi-vector - Multiple Array Vectors (Two registers):
        - SME2 multiple vectors binary int two registers: `ADD`, and `SUB`.
        - SME2 multiple vectors four-way dot product two registers: `SDOT`, and `UDOT`.
        - SME2 multiple vectors long MLA two sources: `SMLAL`, `SMLSL`, `UMLAL`, and `UMLSL`.
        - SME2 multiple vectors long long MLA two sources: `SMLALL`, `SMLSLL`, `UMLALL`, `UMLSLL`, and `USMLALL`.
        - SME2 multiple vectors mixed dot product two registers: `USDOT`.
        - SME2 multiple vectors ternary int two registers: `ADD`, and `SUB`.
        - SME2 multiple vectors two-way dot product two registers: `SDOT`, and `UDOT`.
    - SME2 Multi-vector - Multiple Vectors SVE Destructive (Four registers):
        - SME2 multiple vectors int min/max four registers: `SMAX`, `SMIN`, `UMAX`, and `UMIN`.
    - SME2 Multi-vector - Multiple Vectors SVE Destructive (Two registers):
        - SME2 multiple vectors int min/max two registers: `SMAX`, `SMIN`, `UMAX`, and `UMIN`.
    - SME2 Multi-vector - Multiple Vectors SVE Saturating Multiply (Four registers): `SQDMULH`.
    - SME2 Multi-vector - Multiple Vectors SVE Saturating Multiply (Two registers): `SQDMULH`.
    - SME2 Multi-vector - Multiple and Single Array Vectors (Four registers):
        - SME2 single-multi four-way dot product four registers: `SDOT`, and `UDOT`.
        - SME2 single-multi long MLA four sources: `SMLAL`, `SMLSL`, `UMLAL`, and `UMLSL`.
        - SME2 single-multi long long MLA four sources: `SMLALL`, `SMLSLL`, `SUMLALL`, `UMLALL`, `UMLSLL`, and `USMLALL`.
        - SME2 single-multi mixed dot product four registers: `SUDOT`, and `USDOT`.
        - SME2 single-multi ternary int four registers: `ADD`, and `SUB`.
        - SME2 single-multi two-way dot product four registers: `SDOT`, and `UDOT`.
    - SME2 Multi-vector - Multiple and Single Array Vectors (Two registers):
        - SME2 multiple and single vector long MLA one source: `SMLAL`, `SMLSL`, `UMLAL`, and `UMLSL`.
        - SME2 multiple and single vector long long FMA one source: `SMLALL`, `SMLSLL`, `UMLALL`, `UMLSLL`, and `USMLALL`.
        - SME2 single-multi four-way dot product two registers: `SDOT`, and `UDOT`.
        - SME2 single-multi long MLA two sources: `SMLAL`, `SMLSL`, `UMLAL`, and `UMLSL`.
        - SME2 single-multi long long MLA two sources: `SMLALL`, `SMLSLL`, `SUMLALL`, `UMLALL`, `UMLSLL`, and `USMLALL`.
        - SME2 single-multi mixed dot product two registers: `SUDOT`, and `USDOT`.
        - SME2 single-multi ternary int two registers: `ADD`, and `SUB`.
        - SME2 single-multi two-way dot product two registers: `SDOT`, and `UDOT`.
    - SME2 Multi-vector - Multiple and Single SVE Destructive (Four registers):
        - SME2 single-multi add four registers: `ADD`.
        - SME2 single-multi int min/max four registers: `SMAX`, `SMIN`, `UMAX`, and `UMIN`.
        - SME2 single-multi signed saturating doubling multiply high four registers: `SQDMULH`.
    - SME2 Multi-vector - Multiple and Single SVE Destructive (Two registers):
        - SME2 single-multi add two registers: `ADD`.
        - SME2 single-multi int min/max two registers: `SMAX`, `SMIN`, `UMAX`, and `UMIN`.
        - SME2 single-multi signed saturating doubling multiply high two registers: `SQDMULH`.
    - SME2 Multi-vector - SVE Constructive Binary:
        - SME2 multi-vec CLAMP four registers: `SCLAMP`, and `UCLAMP`.
        - SME2 multi-vec CLAMP two registers: `SCLAMP`, and `UCLAMP`.
        - SME2 multi-vec ZIP two registers: `UZP`, and `ZIP`.
        - SME2 multi-vec quadwords ZIP two registers: `UZP`, and `ZIP`.
    - SME2 Multi-vector - SVE Constructive Unary:
        - SME2 multi-vec ZIP four registers: `UZP`, and `ZIP`.
        - SME2 multi-vec quadwords ZIP four registers: `UZP`, and `ZIP`.
        - SME2 multi-vec unpack four registers: `SUNPK`, and `UUNPK`.
        - SME2 multi-vec unpack two registers: `SUNPK`, and `UUNPK`.
    - SME2 Multi-vector - SVE Select: `SEL`.
    - SME2 Multiple Zero: `ZERO`.
    - SME2 Outer Product - Misc:
        - SME2 32-bit binary outer product: `BMOPA`, and `BMOPS`.
    - SME2 zero lookup table: `ZERO`.
- SVE encodings:
    - SVE Address Generation: `ADR`.
    - SVE Bitwise Immediate: `AND`, `DUPM`, `EOR`, and `ORR`.

- SVE Bitwise Logical - Unpredicated: AND, BCAX, BIC, BSL1N, BSL2N, BSL, EOR3, EOR, NBSL, ORR, and XAR.
- SVE Bitwise Shift - Predicated:
  - SVE bitwise shift by immediate (predicated): ASR, ASRD, LSL, LSR, SRSHR, and URSHR.
  - SVE bitwise shift by vector (predicated): ASR, ASRR, LSL, LSLR, LSR, and LSRR.
  - SVE bitwise shift by wide elements (predicated): ASR, LSL, and LSR.
- SVE Bitwise Shift - Unpredicated: ASR, LSL, and LSR.
- SVE Element Count:
  - SVE element count: CNTB, CNTD, CNTH, and CNTW.
  - SVE inc/dec register by element count: DECB, DECD, DECH, DECW, INCB, INCD, INCH, and INCW.
  - SVE inc/dec vector by element count: DECD, DECH, DECW, INCD, INCH, and INCW.
- SVE Index Generation: INDEX.
- SVE Integer Arithmetic - Unpredicated:
  - SVE integer add/subtract vectors (unpredicated): ADD, and SUB.
- SVE Integer Binary Arithmetic - Predicated:
  - SVE bitwise logical operations (predicated): AND, BIC, EOR, and ORR.
  - SVE integer add/subtract vectors (predicated): ADD, SUB, and SUBR.
  - SVE integer min/max/difference (predicated): SABD, SMAX, SMIN, UABD, UMAX, and UMIN.
  - SVE integer multiply vectors (predicated): MUL, SMULH, and UMULH.
- SVE Integer Compare - Scalars: CTERMEQ, and CTERMNE.
- SVE Integer Compare - Signed Immediate: CMP<CC>.
- SVE Integer Compare - Unsigned Immediate: CMP<CC>.
- SVE Integer Compare - Vectors: CMP<CC>.
- SVE Integer Misc - Unpredicated: MOVPRFX.
- SVE Integer Multiply-Add - Predicated: MAD, MLA, MLS, and MSB.
- SVE Integer Multiply-Add - Unpredicated:
  - SVE integer dot product (unpredicated): SDOT, and UDOT.
  - SVE mixed sign dot product: USDOT.
  - SVE2 complex integer multiply-add: CMLA.
  - SVE2 integer multiply-add long: SMLALB, SMLALT, SMLSLB, SMLSLT, UMLALB, UMLALT, UMLSLB, and UMLSLT.
  - SVE2 saturating multiply-add high: SQRDMLAH.
- SVE Integer Reduction: ADDQV, ANDQV, ANDV, EORQV, EORV, MOVPRFX, ORQV, ORV, SADDV, SMAXQV, SMAXV, SMINQV, SMINV, UADDV, UMAXQV, UMAXV, UMINQV, and UMINV.
- SVE Integer Unary Arithmetic - Predicated:
  - SVE bitwise unary operations (predicated): CLS, CLZ, CNOT, CNT, and NOT.
  - SVE integer unary operations (predicated): ABS, NEG, SXTB, SXTH, SXTW, UXTB, UXTH, and UXTW.
- SVE Integer Wide Immediate - Predicated:
  - SVE copy integer immediate (predicated): CPY.
- SVE Integer Wide Immediate - Unpredicated:
  - SVE broadcast integer immediate (unpredicated): DUP.
  - SVE integer add/subtract immediate (unpredicated): ADD, SUB, and SUBR.
  - SVE integer min/max immediate (unpredicated): SMAX, SMIN, UMAX, and UMIN.
  - SVE integer multiply immediate (unpredicated): MUL.
- SVE Memory - 32-bit Gather and Unsized Contiguous:
  - SVE 32-bit gather load (scalar plus 32-bit unscaled offsets): LD1B, LD1H, LD1SB, LD1SH, and LD1W.
  - SVE 32-bit gather load (vector plus immediate): LD1B, LD1H, LD1SB, LD1SH, and LD1W.
  - SVE 32-bit gather load halfwords (scalar plus 32-bit scaled offsets): LD1H, and LD1SH.
  - SVE 32-bit gather load words (scalar plus 32-bit scaled offsets): LD1W.
  - SVE load and broadcast element: LD1RB, LD1RD, LD1RH, LD1RSB, LD1RSH, LD1RSW, and LD1RW.
  - SVE load predicate register: LDR.
  - SVE load vector register: LDR.
  - SVE2 32-bit gather non-temporal load (vector plus scalar): LDNT1B, LDNT1H, LDNT1SB, LDNT1SH, and LDNT1W.
- SVE Memory - 64-bit Gather:
  - SVE 64-bit gather load (scalar plus 32-bit unpacked scaled offsets): LD1D, LD1H, LD1SH, LD1SW, and LD1W.
  - SVE 64-bit gather load (scalar plus 64-bit scaled offsets): LD1D, LD1H, LD1SH, LD1SW, and LD1W.
  - SVE 64-bit gather load (scalar plus 64-bit unscaled offsets): LD1B, LD1D, LD1H, LD1SB, LD1SH, LD1SW, and LD1W.
  - SVE 64-bit gather load (scalar plus unpacked 32-bit unscaled offsets): LD1B, LD1D, LD1H, LD1SB, LD1SH, LD1SW, and LD1W.
  - SVE 64-bit gather load (vector plus immediate): LD1B, LD1D, LD1H, LD1SB, LD1SH, LD1SW, and LD1W.
  - SVE2 128-bit gather load (vector plus scalar): LD1Q.
  - SVE2 64-bit gather non-temporal load (vector plus scalar): LDNT1B, LDNT1D, LDNT1H, LDNT1SB, LDNT1SH, LDNT1SW, and LDNT1W.
- SVE Memory - Contiguous Load:
  - SVE contiguous load (quadwords, scalar plus immediate): LD1D, and LD1W.
  - SVE contiguous load (quadwords, scalar plus scalar): LD1D, and LD1W.

        - SVE contiguous load (scalar plus immediate): LD1B, LD1D, LD1H, LD1SB, LD1SH, LD1SW, and LD1W.
        - SVE contiguous load (scalar plus scalar): LD1B, LD1D, LD1H, LD1SB, LD1SH, LD1SW, and LD1W.
        - SVE contiguous non-temporal load (scalar plus immediate): LDNT1B, LDNT1D, LDNT1H, and LDNT1W.
        - SVE contiguous non-temporal load (scalar plus scalar): LDNT1B, LDNT1D, LDNT1H, and LDNT1W.
        - SVE load and broadcast quadword (scalar plus immediate): LD1ROB, LD1ROD, LD1ROH, LD1ROW, LD1RQB, LD1RQD, LD1RQH, and LD1RQW.
        - SVE load and broadcast quadword (scalar plus scalar): LD1ROB, LD1ROD, LD1ROH, LD1ROW, LD1RQB, LD1RQD, LD1RQH, and LD1RQW.
        - SVE load multiple structures (quadwords, scalar plus immediate): LD2Q, LD3Q, and LD4Q.
        - SVE load multiple structures (quadwords, scalar plus scalar): LD2Q, LD3Q, and LD4Q.
        - SVE load multiple structures (scalar plus immediate): LD2B, LD2D, LD2H, LD2W, LD3B, LD3D, LD3H, LD3W, LD4B, LD4D, LD4H, and LD4W.
        - SVE load multiple structures (scalar plus scalar): LD2B, LD2D, LD2H, LD2W, LD3B, LD3D, LD3H, LD3W, LD4B, LD4D, LD4H, and LD4W.
    - SVE Memory - Contiguous Store and Unsized Contiguous: ST1B, ST1D, ST1H, ST1W, ST2Q, ST3Q, ST4Q, and STR.
    - SVE Memory - Contiguous Store with Immediate Offset: ST1B, ST1D, ST1H, ST1W, ST2B, ST2D, ST2H, ST2W, ST3B, ST3D, ST3H, ST3W, ST4B, ST4D, ST4H, ST4W, STNT1B, STNT1D, STNT1H, and STNT1W.
    - SVE Memory - Non-temporal and Multi-register Contiguous Store: ST2B, ST2D, ST2H, ST2W, ST3B, ST3D, ST3H, ST3W, ST4B, ST4D, ST4H, ST4W, STNT1B, STNT1D, STNT1H, and STNT1W.
    - SVE Memory - Non-temporal and Quadword Scatter Store: ST1Q, STNT1B, STNT1D, STNT1H, and STNT1W.
    - SVE Memory - Scatter: ST1B, ST1D, ST1H, and ST1W.
    - SVE Memory - Scatter with Optional Sign Extend: ST1B, ST1D, ST1H, and ST1W.
    - SVE Misc: BDEP, BEXT, BGRP, EORBT, EORTB, SADDLBT, SMMLA, SSHLLB, SSHLLT, SSUBLBT, SSUBLTB, UMMLA, USHLLB, USHLLT, and USMMLA.
    - SVE Multiply - Indexed:
        - SVE integer dot product (indexed): SDOT, and UDOT.
        - SVE mixed sign dot product (indexed): SUDOT, and USDOT.
        - SVE2 complex integer multiply-add (indexed): CMLA.
        - SVE2 integer multiply (indexed): MUL.
        - SVE2 integer multiply long (indexed): SMULLB, SMULLT, UMULLB, and UMULLT.
        - SVE2 integer multiply-add (indexed): MLA, and MLS.
        - SVE2 integer multiply-add long (indexed): SMLALB, SMLALT, SMLSLB, SMLSLT, UMLALB, UMLALT, UMLSLB, and UMLSLT.
        - SVE2 saturating multiply high (indexed): SQDMULH, and SQRDMULH.
        - SVE2 saturating multiply-add high (indexed): SQRDMLAH.
    - SVE Permute Predicate:
        - SVE permute predicate elements: TRN1, TRN2, UZP1, UZP2, ZIP1, and ZIP2.
        - SVE reverse predicate elements: REV.
    - SVE Permute Vector - Extract: EXT.
    - SVE Permute Vector - Indexed DUP: DUP.
    - SVE Permute Vector - Interleaving: TRN1, TRN2, UZP1, UZP2, ZIP1, and ZIP2.
    - SVE Permute Vector - One Source Quadwords: DUPQ, and EXTQ.
    - SVE Permute Vector - Predicated:
        - SVE copy SIMD&FP scalar register to vector (predicated): CPY.
        - SVE copy general register to vector (predicated): CPY.
        - SVE reverse doublewords: REVD.
        - SVE reverse within elements: RBIT, REVB, REVH, and REVW.
    - SVE Permute Vector - Segments: TRN1, TRN2, UZP1, UZP2, ZIP1, and ZIP2.
    - SVE Permute Vector - TBXQ: TBXQ.
    - SVE Permute Vector - Three Sources TBL: TBL, and TBX.
    - SVE Permute Vector - Two Sources Quadwords: TBLQ, UZPQ1, UZPQ2, ZIPQ1, and ZIPQ2.
    - SVE Permute Vector - Two Sources TBL: TBL.
    - SVE Permute Vector - Unpredicated:
        - SVE broadcast general register: DUP.
        - SVE insert SIMD&FP scalar register: INSR.
        - SVE insert general register: INSR.
        - SVE reverse vector elements: REV.
        - SVE unpack vector elements: SUNPKHI, SUNPKLO, UUNPKHI, and UUNPKLO.
    - SVE Predicate Logical Operations: AND, ANDS, BIC, BICS, EOR, EORS, NAND, NANDS, NOR, NORS, ORN, ORNS, ORR, ORRS, and SEL.
    - SVE Predicate Misc:
        - SVE predicate initialize: PTRUE, and PTRUES.
        - SVE predicate read from FFR (predicated): RDFFR, and RDFFRS.
        - SVE predicate read from FFR (unpredicated): RDFFR.
        - SVE predicate test: PTEST.
        - SVE predicate zero: PFALSE.

- SVE Predicate Select: PSEL.
- SVE Scalar Integer Compare - Predicate-as-counter: PTRUE.
- SVE Stack Allocation: ADDPL, ADDSPL, ADDSVL, ADDVL, RDSVL, and RDVL.
- SVE Vector Select: SEL.
- SVE Write FFR: SETFFR, and WRFFR.
- SVE integer clamp: SCLAMP, and UCLAMP.
- SVE two-way dot product: SDOT, and UDOT.
- SVE two-way dot product (indexed): SDOT, and UDOT.
- SVE2 Accumulate:
  - SVE2 bitwise shift and insert: SLI, and SRI.
  - SVE2 bitwise shift right and accumulate: SRSRA, SSRA, URSRA, and USRA.
  - SVE2 complex integer add: CADD.
  - SVE2 integer absolute difference and accumulate: SABA, and UABA.
  - SVE2 integer absolute difference and accumulate long: SABALB, SABALT, UABALB, and UABALT.
  - SVE2 integer add/subtract long with carry: ADCLB, ADCLT, SBCLB, and SBCLT.
- SVE2 Crypto Extensions: AESD, AESE, AESIMC, AESMC, RAX1, SM4E, and SM4EKEY.
- SVE2 Histogram Computation (Segment) and Lookup Table: LUTI2, and LUTI4.
- SVE2 Integer - Predicated:
  - SVE2 integer halving add/subtract (predicated): SHADD, SHSUB, SHSUBR, UHADD, UHSUB ,and UHSUBR.
  - SVE2 integer pairwise add and accumulate long: SADALP, and UADALP.
  - SVE2 integer pairwise arithmetic: ADDP, SMAXP, SMINP, UMAXP, and UMINP.
- SVE2 Integer Multiply - Unpredicated: MUL, PMUL, SMULH, SQDMULH, SQRDMULH, and UMULH.
- SVE2 Narrowing:
  - SVE2 bitwise shift right narrow: RSHRNB, RSHRNT, SHRNB, and SHRNT.
  - SVE2 integer add/subtract narrow high part: ADDHNB, ADDHNT, RADDHNB, RADDHNT, RSUBHNB, RSUBHNT, SUBHNB, and SUBHNT.
- SVE2 Widening Integer Arithmetic:
  - SVE2 integer add/subtract long: SABDLB, SABDLT, SADDLB, SADDLT, SSUBLB, SSUBLT, UABDLB, UABDLT, UADDLB, UADDLT, USUBLB, and USUBLT.
  - SVE2 integer add/subtract wide: SADDWB, SADDWT, SSUBWB, SSUBWT, UADDWB, UADDWT, USUBWB, and USUBWT.
  - SVE2 integer multiply long: PMULLB, PMULLT, SMULLB, SMULLT, UMULLB, and UMULLT.

Note

The architecture makes no statement about the timing properties when the PSTATE.DIT bit is not set. However, it is likely that many of these instructions have timing that is invariant of the data in many situations.

In particular, Arm strongly recommends that the Armv8.3 pointer authentication instructions do not have their timing dependent on the key value used in the pointer authentication in all cases, regardless of the PSTATE.DIT bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [23:0]

Reserved, RES0.

Accessing DIT

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DIT
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_DIT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    X[t, 64] = Zeros(39):PSTATE.DIT:Zeros(24);
elsif PSTATE.EL == EL1 then
    X[t, 64] = Zeros(39):PSTATE.DIT:Zeros(24);
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(39):PSTATE.DIT:Zeros(24);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(39):PSTATE.DIT:Zeros(24);
```

MSR DIT, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_DIT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    PSTATE.DIT = X[t, 64]<24>;
elsif PSTATE.EL == EL1 then
    PSTATE.DIT = X[t, 64]<24>;
elsif PSTATE.EL == EL2 then
    PSTATE.DIT = X[t, 64]<24>;
elsif PSTATE.EL == EL3 then
    PSTATE.DIT = X[t, 64]<24>;
```

MSR DIT, #<imm>

op0	op1	CRn	op2
0b00	0b011	0b0100	0b010

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DLR_EL0, Debug Link Register

The DLR_EL0 characteristics are:

Purpose

In Debug state, holds the address to restart from.

Configuration

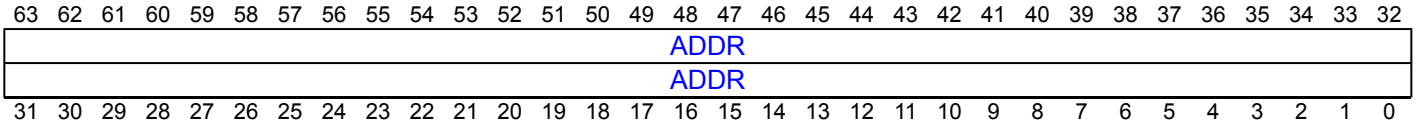
AArch64 System register DLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DLR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DLR_EL0 are UNDEFINED.

Attributes

DLR_EL0 is a 64-bit register.

Field descriptions



ADDR, bits [63:0]

Restart address.

Accessing DLR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DLR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    X[t, 64] = DLR_EL0;
```

```
MSR DLR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    DLR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DSPSR_EL0, Debug Saved Program Status Register

The DSPSR_EL0 characteristics are:

Purpose

Holds the saved process state for Debug state. On entering Debug state, PSTATE information is written to this register. On exiting Debug state, values are copied from this register to PSTATE.

Configuration

AArch64 System register DSPSR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DSPSR[31:0].

AArch64 System register DSPSR_EL0 bits [63:32] are architecturally mapped to AArch32 System register DSPSR2[31:0] when FEAT_Debugv8p9 is implemented.

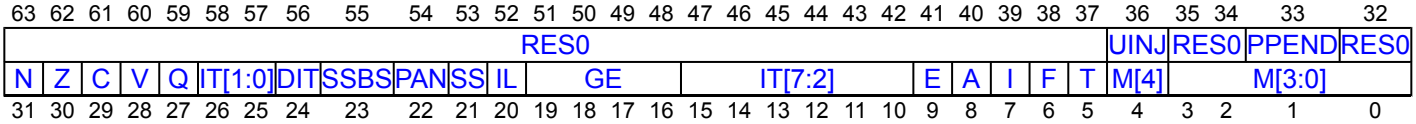
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DSPSR_EL0 are UNDEFINED.

Attributes

DSPSR_EL0 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented and exiting Debug state to AArch32 state:



Bits [63:37]

Reserved, RES0.

UINJ, bit [36]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Copied to PSTATE.UINJ on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:34]

Reserved, RES0.

PPEND, bit [33]

When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Copied to PSTATE.PPEND on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Copied to PSTATE.N on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Copied to PSTATE.Z on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Copied to PSTATE.C on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Copied to PSTATE.V on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Copied to PSTATE.Q on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Copied to PSTATE.IT on exiting Debug state.

On exiting Debug state, DSPSR_EL0.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is DSPSR_EL0[26:25].
- IT[7:2] is DSPSR_EL0[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Copied to PSTATE.DIT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Copied to PSTATE.SSBS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Copied to PSTATE.PAN on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Copied to PSTATE.SS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Copied to PSTATE.IL on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Copied to PSTATE.GE on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Copied to PSTATE.E on exiting Debug state.

If the implementation does not support big-endian operation, DSPSR_EL0.E is RES0. If the implementation does not support little-endian operation, DSPSR_EL0.E is RES1. On exiting Debug state, if the implementation does not support big-endian operation at the Exception level being returned to, DSPSR_EL0.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, DSPSR_EL0.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Copied to PSTATE.A on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Copied to PSTATE.I on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Copied to PSTATE.F on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Copied to PSTATE.T on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Copied to PSTATE.nRW on exiting Debug state.

M[4]	Meaning
0b1	AArch32 execution state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch32 Mode. Copied to PSTATE.M[3:0] on exiting Debug state.

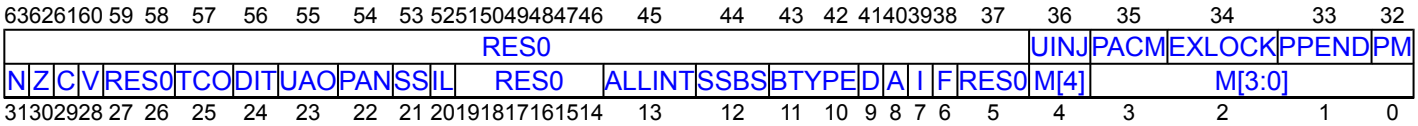
M[3:0]	Meaning
0b0000	User.
0b0001	FIQ.
0b0010	IRQ.
0b0011	Supervisor.
0b0110	Monitor.
0b0111	Abort.
0b1010	Hyp.
0b1011	Undefined.
0b1111	System.

Other values are reserved. If DSPSR_EL0.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_AA64 is implemented and entering or exiting Debug state from or to AArch64 state:



Bits [63:37]

Reserved, RES0.

UINJ, bit [36]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Set to the value of PSTATE.UINJ on entering Debug state, and copied to PSTATE.UINJ on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PACM, bit [35]

When FEAT_PAuth_LR is implemented:

PACM. Set to the value of PSTATE.PACM on entering Debug state, and copied to PSTATE.PACM on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLOCK, bit [34]
When FEAT_GCS is implemented:

Exception return state lock. Set to the value of PSTATE.EXLOCK on entering Debug state, and copied to PSTATE.EXLOCK on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PPEND, bit [33]
When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Set to the value of PSTATE.PPEND on entering Debug state, and conditionally copied to PSTATE.PPEND on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PM, bit [32]
When FEAT_EBEP is implemented:

Profiling exception mask bit. Set to the value of PSTATE.PM on entering Debug state, and copied to PSTATE.PM on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on entering Debug state, and copied to PSTATE.N on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on entering Debug state, and copied to PSTATE.Z on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on entering Debug state, and copied to PSTATE.C on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on entering Debug state, and copied to PSTATE.V on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]

When FEAT_MTE is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on entering Debug state, and copied to PSTATE.TCO on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on entering Debug state, and copied to PSTATE.DIT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]

When FEAT_UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on entering Debug state, and copied to PSTATE.UAO on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on entering Debug state, and copied to PSTATE.PAN on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on entering Debug state, and conditionally copied to PSTATE.SS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on entering Debug state, and copied to PSTATE.IL on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

ALLINT, bit [13]

When FEAT_NMI is implemented:

All IRQ or FIQ interrupts mask. Set to the value of PSTATE.ALLINT on entering Debug state, and copied to PSTATE.ALLINT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [12]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on entering Debug state, and copied to PSTATE.SSBS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]

When FEAT_BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on entering Debug state, and copied to PSTATE.BTYPE on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on entering Debug state, and copied to PSTATE.D on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on entering Debug state, and copied to PSTATE.A on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on entering Debug state, and copied to PSTATE.I on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on entering Debug state, and copied to PSTATE.F on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to 0b0, the value of PSTATE.nRW, on entering Debug state from AArch64 state, and copied to PSTATE.nRW on exiting Debug state.

M[4]	Meaning
0b0	AArch64 execution state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

M[3:0]	Meaning
0b0000	EL0.
0b0100	EL1 with SP_EL0 (EL1t).
0b0101	EL1 with SP_EL1 (EL1h).
0b1000	EL2 with SP_EL0 (EL2t).
0b1001	EL2 with SP_EL2 (EL2h).
0b1100	EL3 with SP_EL0 (EL3t).
0b1101	EL3 with SP_EL3 (EL3h).

Other values are reserved. If DSPSR_EL0.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on entering Debug state and copied to PSTATE.EL on exiting Debug state.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on entering Debug state and copied to PSTATE.SP on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DSPSR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DSPSR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    X[t, 64] = DSPSR_EL0;
```

```
MSR DSPSR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2

0b11	0b011	0b0100	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    DSPSR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DVP RCTX, Data Value Prediction Restriction by Context

The DVP RCTX characteristics are:

Purpose

Data Value Prediction Restriction by Context applies to all Data Value Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

Note

The prediction of the PSTATE.{N,Z,C,V} values is not considered a data value for this purpose.

Data value predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control speculative execution occurring after the instruction is complete and synchronized.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

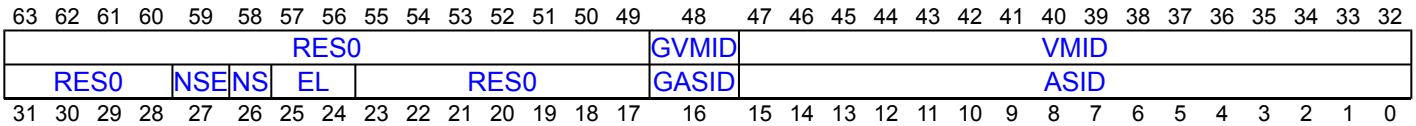
Configuration

This instruction is present only when FEAT_SPECRES is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DVP RCTX are UNDEFINED.

Attributes

DVP RCTX is a 64-bit System instruction.

Field descriptions



Bits [63:49]

Reserved, RES0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 target execution context.
0b1	Applies to all VMIDs for an EL0 or EL1 target execution context.

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and the target execution context is either:

- EL1.
- EL0 when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, this field is treated as the current VMID.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

If the implementation supports 16 bits of VMID, then the upper 8 bits of the VMID must be written to 0 by software when the context being affected only uses 8 bits.

Bits [31:28]

Reserved, RES0.

NSE, bit [27]

When FEAT_RME is implemented:

Together with the NS field, selects the Security state.

For a description of the values derived by evaluating NS and NSE together, see DVP_RCTX.NS.

Otherwise:

Reserved, RES0.

NS, bit [26]

When FEAT_RME is implemented:

Together with the NSE field, selects the Security state.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

Some Effective values are determined by the current Security state:

- When executed in Secure state, the Effective value of NSE is 0.
- When executed in Non-secure state, the Effective value of {NSE, NS} is {0, 1}.
- When executed in Realm state, the Effective value of {NSE, NS} is {1, 1}.

This instruction is treated as a NOP when executed at EL3 and either:

- DVP_RCTX.{NSE, NS} selects a reserved value.
- DVP_RCTX.{NSE, NS} == {1, 0} and DVP_RCTX.EL has a value other than 0b11.

Otherwise:

Security State.

NS	Meaning
0b0	Secure state.
0b1	Non-secure state.

When executed in Non-secure state, the Effective value of NS is 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

If the instruction is executed at an Exception level lower than the specified level, or is specified to apply to a combination of Exception level and Security state that is not implemented, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

GASID	Meaning
0b0	Applies to specified ASID for an EL0 target execution context.
0b1	Applies to all ASIDs for an EL0 target execution context.

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [15:0]

Only applies for an EL0 target execution context and when bit[16] is 0.

Otherwise this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being affected only uses 8 bits.

Executing DVP RCTX

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
DVP RCTX, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0011	0b101

```
if !(IsFeatureImplemented(FEAT_SPECRES) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DVPRCTX == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnRCTX == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.RestrictPrediction(X[t, 64], RestrictType_DataValue);
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.DVPRCTX == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.RestrictPrediction(X[t, 64], RestrictType_DataValue);
elsif PSTATE.EL == EL2 then
    AArch64.RestrictPrediction(X[t, 64], RestrictType_DataValue);
elsif PSTATE.EL == EL3 then
    AArch64.RestrictPrediction(X[t, 64], RestrictType_DataValue);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ELR_EL1, Exception Link Register (EL1)

The ELR_EL1 characteristics are:

Purpose

When taking an exception to EL1, holds the address to return to.

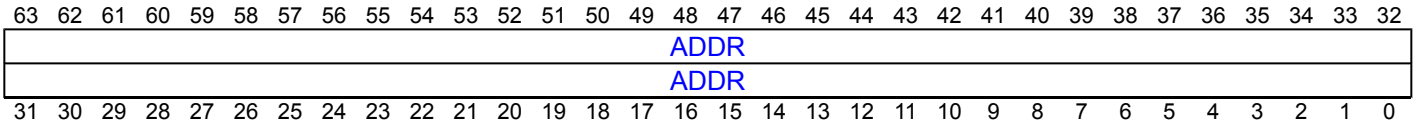
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ELR_EL1 are UNDEFINED.

Attributes

ELR_EL1 is a 64-bit register.

Field descriptions



ADDR, bits [63:0]

Return address.

An exception return from EL1 using AArch64 makes ELR_EL1 become UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ELR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name ELR_EL1 or ELR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ELR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x230];
    else
        X[t, 64] = ELR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = ELR_EL2;
    else
        X[t, 64] = ELR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ELR_EL1;
```

```
MSR ELR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& !(EffectiveHCR_EL2_NVx() IN {'x11'}) then
        EXLOCKException();
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x230] = X[t, 64];
    else
        ELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& ELIsInHost(EL2) then
        EXLOCKException();
    elsif ELIsInHost(EL2) then
        ELR_EL2 = X[t, 64];
    else
        ELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ELR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, ELR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x230];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = ELR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = ELR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR ELR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x230] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        ELR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        ELR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MRS <Xt>, ELR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = ELR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ELR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ELR_EL2;
```

When FEAT_VHE is implemented

```
MSR ELR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& EffectiveHCR_EL2_NVx() IN {'xx1'} then
        EXLOCKException();
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        ELR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
then
        EXLOCKException();
    else
        ELR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ELR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ELR_EL2, Exception Link Register (EL2)

The ELR_EL2 characteristics are:

Purpose

When taking an exception to EL2, holds the address to return to.

Configuration

AArch64 System register ELR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ELR_hyp[31:0].

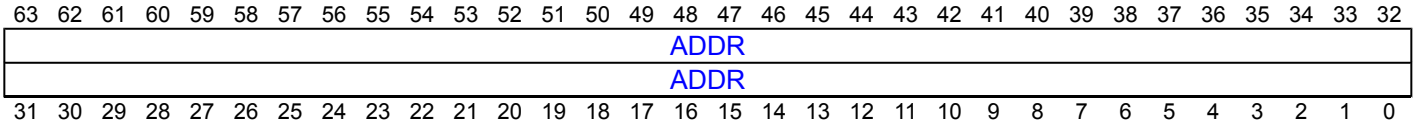
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ELR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ELR_EL2 is a 64-bit register.

Field descriptions



ADDR, bits [63:0]

Return address.

An exception return from EL2 using AArch64 makes ELR_EL2 become UNKNOWN.

When EL2 is in AArch32 Execution state and an exception is taken from EL0, EL1, or EL2 to EL3 and AArch64 execution, the upper 32-bits of ELR_EL2 are either set to 0 or hold the same value that they did before AArch32 execution. Which option is adopted is determined by an implementation, and might vary dynamically within an implementation. Correspondingly software must regard the value as being an UNKNOWN choice between the two values.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ELR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name ELR_EL2 or ELR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ELR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = ELR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ELR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ELR_EL2;
```

```
MSR ELR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& EffectiveHCR_EL2_NVx() IN {'xx1'} then
        EXLOCKException();
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        ELR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
then
        EXLOCKException();
    else
        ELR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ELR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, ELR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x230];
    else
        X[t, 64] = ELR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = ELR_EL2;
    else
        X[t, 64] = ELR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ELR_EL1;
```

When FEAT_VHE is implemented

```
MSR ELR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& !(EffectiveHCR_EL2_NVx() IN {'x11'}) then
        EXLOCKException();
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x230] = X[t, 64];
    else
        ELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& ELIsInHost(EL2) then
        EXLOCKException();
    elsif ELIsInHost(EL2) then
        ELR_EL2 = X[t, 64];
    else
        ELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ELR_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ELR_EL3, Exception Link Register (EL3)

The ELR_EL3 characteristics are:

Purpose

When taking an exception to EL3, holds the address to return to.

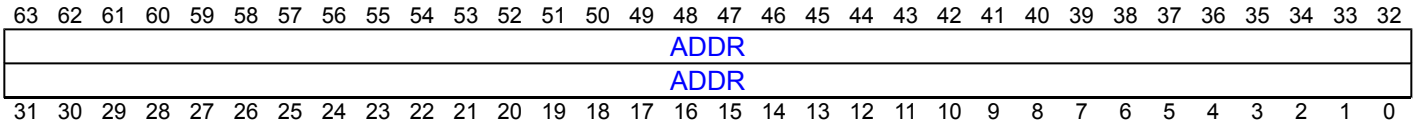
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ELR_EL3 are UNDEFINED.

Attributes

ELR_EL3 is a 64-bit register.

Field descriptions



ADDR, bits [63:0]

Return address.

An exception return from EL3 using AArch64 makes ELR_EL3 become UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ELR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ELR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0100	0b0000	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ELR_EL3;
```

```
MSR ELR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0100	0b0000	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
then
        EXLOCKException();
    else
        ELR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRIDR_EL1, Error Record ID Register

The ERRIDR_EL1 characteristics are:

Purpose

Defines the highest numbered index of the error records that can be accessed through the Error Record System registers.

Configuration

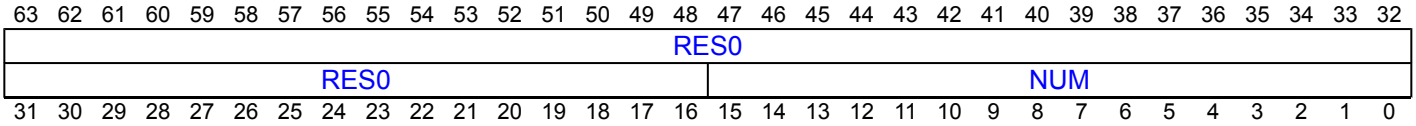
AArch64 System register ERRIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERRIDR[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERRIDR_EL1 are UNDEFINED.

Attributes

ERRIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

NUM, bits [15:0]

Highest numbered index of the records that can be accessed through the Error Record System registers plus one. Zero indicates no records can be accessed through the Error Record System registers.

Each implemented record is owned by a node. A node might own multiple records.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERRIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERRIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERRIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERRIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERRIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRSELR_EL1, Error Record Select Register

The ERRSELR_EL1 characteristics are:

Purpose

Selects an error record to be accessed through the Error Record System registers.

Configuration

AArch64 System register ERRSELR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERRSELR[31:0].

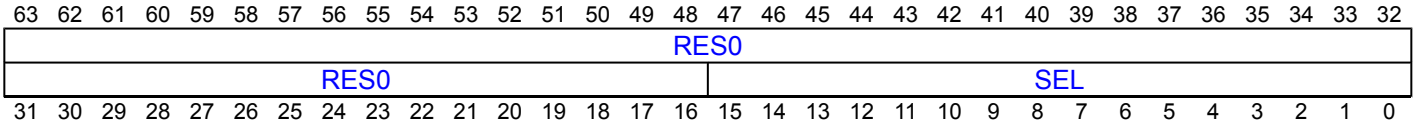
This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERRSELR_EL1 are UNDEFINED.

If ERRIDR_EL1 indicates that zero error records are implemented, then it is IMPLEMENTATION DEFINED whether ERRSELR_EL1 is UNDEFINED or RES0.

Attributes

ERRSELR_EL1 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

SEL, bits [15:0]

Selects the error record accessed through the ERX registers.

For example, if ERRSELR_EL1.SEL is `0x0004`, then direct reads and writes of ERXSTATUS_EL1 access ERR4STATUS.

If ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then all of the following apply:

- The value read back from ERRSELR_EL1.SEL is UNKNOWN.
- One of the following occurs:
  - An UNKNOWN error record is selected.
  - The ERX*_EL1 registers are RAZ/WI.
  - ERX*_EL1 register reads and writes are NOPs.
  - ERX*_EL1 register reads and writes are UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ERRSELR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERRSELR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERRSELR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERRSELR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERRSELR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERRSELR_EL1;
```

MSR ERRSELR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ERRSELR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERRSELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERRSELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERRSELR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXADDR_EL1, Selected Error Record Address Register

The ERXADDR_EL1 characteristics are:

Purpose

Accesses ERR<n>ADDR for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXADDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXADDR[31:0].

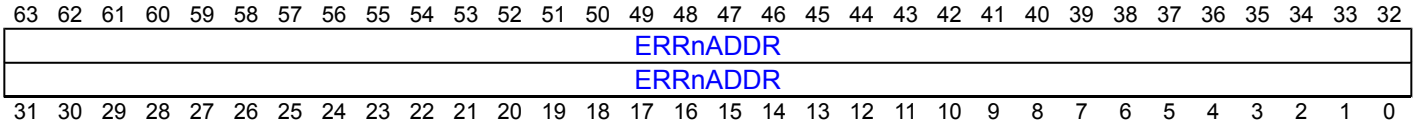
AArch64 System register ERXADDR_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXADDR2[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXADDR_EL1 are UNDEFINED.

Attributes

ERXADDR_EL1 is a 64-bit register.

Field descriptions



ERRnADDR, bits [63:0]

ERXADDR_EL1 accesses ERR<n>ADDR, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXADDR_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXADDR_EL1 is RAZ/WI.
- Direct reads and writes of ERXADDR_EL1 are NOPs.
- Direct reads and writes of ERXADDR_EL1 are UNDEFINED.

ERR<n>ADDR describes additional constraints that also apply when ERR<n>ADDR is accessed through ERXADDR_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXADDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b011

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXADDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXADDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXADDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXADDR_EL1;
```

```
MSR ERXADDR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b011

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ERXADDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXADDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXADDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXADDR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXCTLR_EL1, Selected Error Record Control Register

The ERXCTLR_EL1 characteristics are:

Purpose

Accesses ERR<n>CTLR for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXCTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXCTLR[31:0].

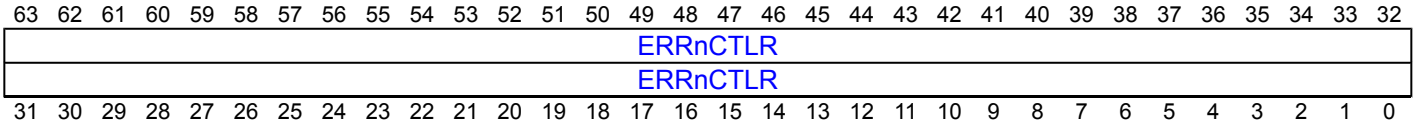
AArch64 System register ERXCTLR_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXCTLR2[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXCTLR_EL1 are UNDEFINED.

Attributes

ERXCTLR_EL1 is a 64-bit register.

Field descriptions



ERRnCTLR, bits [63:0]

ERXCTLR_EL1 accesses ERR<n>CTLR, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXCTLR_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXCTLR_EL1 is RAZ/WI.
- Direct reads and writes of ERXCTLR_EL1 are NOPs.
- Direct reads and writes of ERXCTLR_EL1 are UNDEFINED.

If ERRSELR_EL1.SEL is not the index of the first error record owned by a node, then ERR<n>CTLR is not present, meaning reads and writes of ERXCTLR_EL1 are RES0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXCTLR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXCTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXCTLR_EL1;
```

```
MSR ERXCTLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ERXCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXCTLR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXFR_EL1, Selected Error Record Feature Register

The ERXFR_EL1 characteristics are:

Purpose

Accesses ERR<n>FR for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXFR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXFR[31:0].

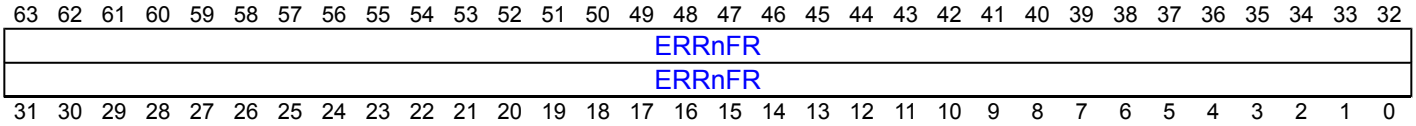
AArch64 System register ERXFR_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXFR2[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXFR_EL1 are UNDEFINED.

Attributes

ERXFR_EL1 is a 64-bit register.

Field descriptions



ERRnFR, bits [63:0]

ERXFR_EL1 accesses ERR<n>FR, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXFR_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXFR_EL1 is RAZ.
- Direct reads of ERXFR_EL1 are NOPs.
- Direct reads of ERXFR_EL1 are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXFR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXFR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXFR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXGSR_EL1, Selected Error Record Group Status Register

The ERXGSR_EL1 characteristics are:

Purpose

Shows the status for the records in a group of error records.

Accesses ERRGSR for the group of error records <n> selected by ERRSELR_EL1.SEL[15:6].

Configuration

This register is present only when FEAT_RASv2 is implemented. Otherwise, direct accesses to ERXGSR_EL1 are UNDEFINED.

Attributes

ERXGSR_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
S63	S62	S61	S60	S59	S58	S57	S56	S55	S54	S53	S52	S51	S50	S49	S48	S47	S46	S45	S44	S43	S42	S41	S40	S39	S38	S37	S36	S35	S34	S33
S31	S30	S29	S28	S27	S26	S25	S24	S23	S22	S21	S20	S19	S18	S17	S16	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

S<q>, bit [q], for q = 63 to 0

When error record m is implemented and error record m supports this type of reporting:

The status for error record <m>, where m = q + (UInt(ERRSELR_EL1.SEL[15:6])×64). A read-only copy of ERR<m>STATUS.V.

S<q>	Meaning
0b0	No error.
0b1	One or more errors.

Otherwise:

Reserved, RES0.

Accessing ERXGSR_EL1

If ERRIDR_EL1.NUM is 0x0000 or ERRSELR_EL1.SEL[15:6] is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN group of error records are selected.
- ERXGSR_EL1 is RAZ.
- Direct reads of ERXGSR_EL1 are NOPs.
- Direct reads of ERXGSR_EL1 are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXGSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_RASv2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nERXGSR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXGSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXGSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXGSR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0

The ERXMISC0_EL1 characteristics are:

Purpose

Accesses ERR<n>MISC0 for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXMISC0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC0[31:0].

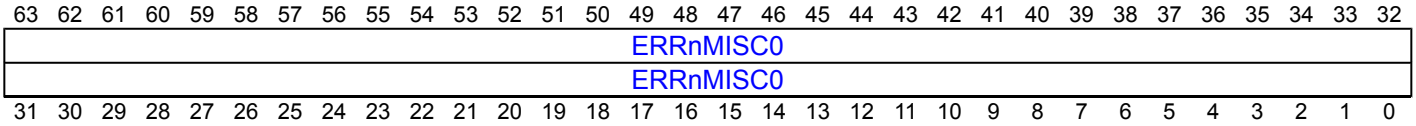
AArch64 System register ERXMISC0_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC1[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXMISC0_EL1 are UNDEFINED.

Attributes

ERXMISC0_EL1 is a 64-bit register.

Field descriptions



ERRnMISC0, bits [63:0]

ERXMISC0_EL1 accesses ERR<n>MISC0, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXMISC0_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC0_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC0_EL1 are NOPs.
- Direct reads and writes of ERXMISC0_EL1 are UNDEFINED.

ERR<n>MISC0 describes additional constraints that also apply when ERR<n>MISC0 is accessed through ERXMISC0_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXMISC0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXMISC0_EL1;
```

```
MSR ERXMISC0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXMISC0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1

The ERXMISC1_EL1 characteristics are:

Purpose

Accesses ERR<n>MISC1 for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXMISC1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC2[31:0].

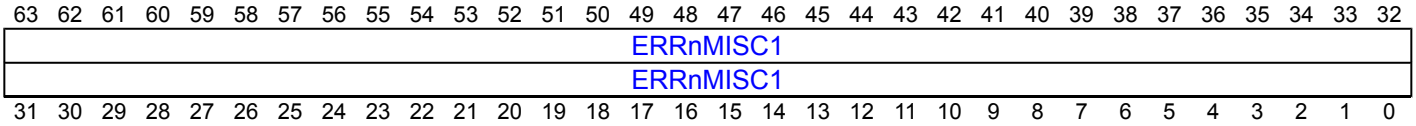
AArch64 System register ERXMISC1_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC3[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXMISC1_EL1 are UNDEFINED.

Attributes

ERXMISC1_EL1 is a 64-bit register.

Field descriptions



ERRnMISC1, bits [63:0]

ERXMISC1_EL1 accesses ERR<n>MISC1, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXMISC1_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC1_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC1_EL1 are NOPs.
- Direct reads and writes of ERXMISC1_EL1 are UNDEFINED.

ERR<n>MISC1 describes additional constraints that also apply when ERR<n>MISC1 is accessed through ERXMISC1_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXMISC1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXMISC1_EL1;
```

```
MSR ERXMISC1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXMISC1_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC2_EL1, Selected Error Record Miscellaneous Register 2

The ERXMISC2_EL1 characteristics are:

Purpose

Accesses ERR<n>MISC2 for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXMISC2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC4[31:0].

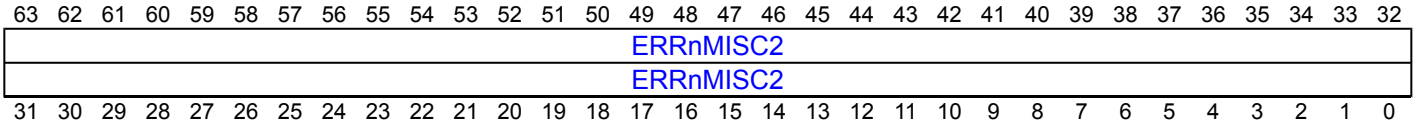
AArch64 System register ERXMISC2_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC5[31:0].

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXMISC2_EL1 are UNDEFINED.

Attributes

ERXMISC2_EL1 is a 64-bit register.

Field descriptions



ERRnMISC2, bits [63:0]

ERXMISC2_EL1 accesses ERR<n>MISC2, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXMISC2_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC2_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC2_EL1 are NOPs.
- Direct reads and writes of ERXMISC2_EL1 are UNDEFINED.

ERR<n>MISC2 describes additional constraints that also apply when ERR<n>MISC2 is accessed through ERXMISC2_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXMISC2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b010

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXMISC2_EL1;
```

```
MSR ERXMISC2_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b010

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXMISC2_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC3_EL1, Selected Error Record Miscellaneous Register 3

The ERXMISC3_EL1 characteristics are:

Purpose

Accesses ERR<n>MISC3 for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXMISC3_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC6[31:0].

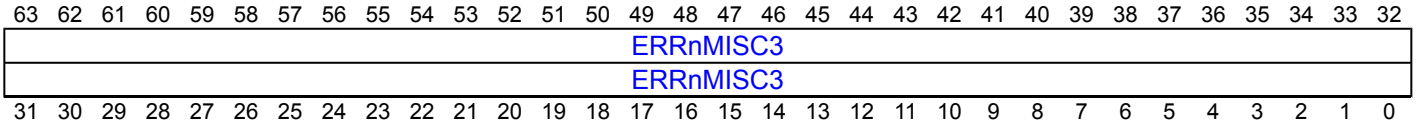
AArch64 System register ERXMISC3_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC7[31:0].

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXMISC3_EL1 are UNDEFINED.

Attributes

ERXMISC3_EL1 is a 64-bit register.

Field descriptions



ERRnMISC3, bits [63:0]

ERXMISC3_EL1 accesses ERR<n>MISC3, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXMISC3_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC3_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC3_EL1 are NOPs.
- Direct reads and writes of ERXMISC3_EL1 are UNDEFINED.

ERR<n>MISC3 describes additional constraints that also apply when ERR<n>MISC3 is accessed through ERXMISC3_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXMISC3_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b011

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC3_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC3_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXMISC3_EL1;
```

```
MSR ERXMISC3_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b011

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC3_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC3_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXMISC3_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXPFGCDN_EL1, Selected Pseudo-fault Generation Countdown Register

The ERXPFGCDN_EL1 characteristics are:

Purpose

Accesses ERR<n>PFGCDN for the error record <n> selected by ERRSELR_EL1.SEL.

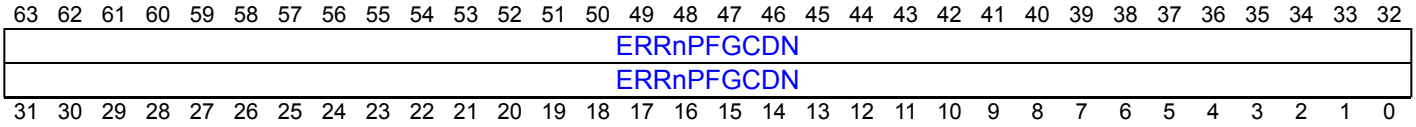
Configuration

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXPFGCDN_EL1 are UNDEFINED.

Attributes

ERXPFGCDN_EL1 is a 64-bit register.

Field descriptions



ERRnPFGCDN, bits [63:0]

ERXPFGCDN_EL1 accesses ERR<n>PFGCDN, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXPFGCDN_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXPFGCDN_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCDN_EL1 are NOPs.
- Direct reads and writes of ERXPFGCDN_EL1 are UNDEFINED.

If ERRSELR_EL1.SEL selects an error record owned by a node that does not implement the Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGCDN_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCDN_EL1 are NOPs.
- Direct reads and writes of ERXPFGCDN_EL1 are UNDEFINED.

Note

A node does not implement the Common Fault Injection Model Extension if ERR<q>FR.INJ reads as `0b00`. <q> is the index of the first error record owned by the same node as error record <n>, where <n> is the value in ERRSELR_EL1.SEL. If the node owns a single record then q = n.

If ERRSELR_EL1.SEL is not the index of the first error record owned by a node, then ERR<n>PFGCDN is not present, meaning reads and writes of ERXPFGCDN_EL1 are RES0.

ERR<n>PFGCDN describes additional constraints that also apply when ERR<n>PFGCDN is accessed through ERXPFGCDN_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXPFGCDN_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b110

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXPFGCDN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGCDN_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGCDN_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXPFGCDN_EL1;
```

```
MSR ERXPFGCDN_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b110

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ERXPFGCDN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXPFGCDN_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXPFGCDN_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXPFGCDN_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXPFGCTL_EL1, Selected Pseudo-fault Generation Control Register

The ERXPFGCTL_EL1 characteristics are:

Purpose

Accesses ERR<n>PFGCTL for the error record <n> selected by ERRSELR_EL1.SEL.

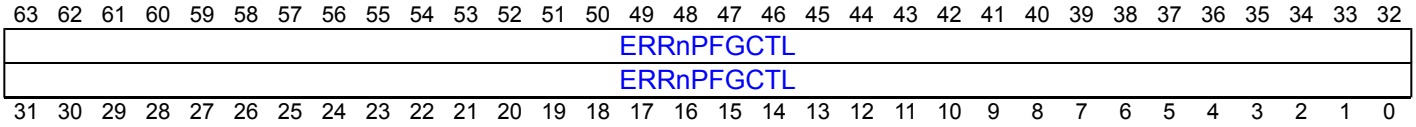
Configuration

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXPFGCTL_EL1 are UNDEFINED.

Attributes

ERXPFGCTL_EL1 is a 64-bit register.

Field descriptions



ERRnPFGCTL, bits [63:0]

ERXPFGCTL_EL1 accesses ERR<n>PFGCTL, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXPFGCTL_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXPFGCTL_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCTL_EL1 are NOPs.
- Direct reads and writes of ERXPFGCTL_EL1 are UNDEFINED.

If ERRSELR_EL1.SEL selects an error record owned by a node that does not implement the Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGCTL_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCTL_EL1 are NOPs.
- Direct reads and writes of ERXPFGCTL_EL1 are UNDEFINED.

Note

A node does not implement the Common Fault Injection Model Extension if ERR<q>FR.INJ reads as `0b00`. <q> is the index of the first error record owned by the same node as error record <n>, where <n> is the value in ERRSELR_EL1.SEL. If the node owns a single record then q = n.

If ERRSELR_EL1.SEL is not the index of the first error record owned by a node, then ERR<n>PFGCTL is not present, meaning reads and writes of ERXPFGCTL_EL1 are RES0.

ERR<n>PFGCTL describes additional constraints that also apply when ERR<n>PFGCTL is accessed through ERXPFGCTL_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXPFGCTL_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXPFGCTL_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGCTL_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGCTL_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXPFGCTL_EL1;
```

```
MSR ERXPFGCTL_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ERXPFGCTL_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXPFGCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXPFGCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXPFGCTL_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXPFGF_EL1, Selected Pseudo-fault Generation Feature Register

The ERXPFGF_EL1 characteristics are:

Purpose

Accesses ERR<n>PFGF for the error record <n> selected by ERRSELR_EL1.SEL.

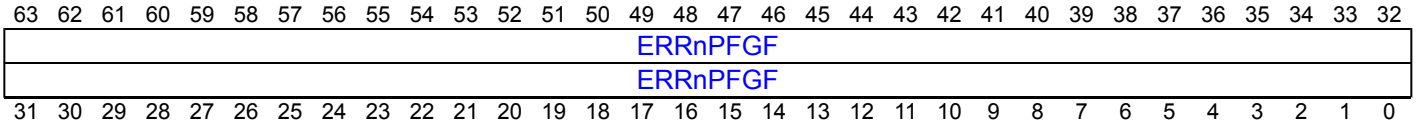
Configuration

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXPFGF_EL1 are UNDEFINED.

Attributes

ERXPFGF_EL1 is a 64-bit register.

Field descriptions



ERRnPFGF, bits [63:0]

ERXPFGF_EL1 accesses ERR<n>PFGF, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXPFGF_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXPFGF_EL1 is RAZ.
- Direct reads of ERXPFGF_EL1 are NOPs.
- Direct reads of ERXPFGF_EL1 are UNDEFINED.

If ERRSELR_EL1.SEL selects an error record owned by a node that does not implement the Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGF_EL1 is RAZ.
- Direct reads of ERXPFGF_EL1 are NOPs.
- Direct reads of ERXPFGF_EL1 are UNDEFINED.

Note

A node does not implement the Common Fault Injection Model Extension if ERR<q>FR.INJ reads as `0b00`. <q> is the index of the first error record owned by the same node as error record <n>, where <n> is the value in ERRSELR_EL1.SEL. If the node owns a single record then q = n.

If ERRSELR_EL1.SEL is not the index of the first error record owned by a node, then ERR<n>PFGF is not present, meaning reads of ERXPFGF_EL1 are RES0.

ERR<n>PFGF describes additional constraints that also apply when ERR<n>PFGF is accessed through ERXPFGF_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXPFGF_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXPFGF_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGF_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGF_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXPFGF_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXSTATUS_EL1, Selected Error Record Primary Status Register

The ERXSTATUS_EL1 characteristics are:

Purpose

Accesses ERR<n>STATUS for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

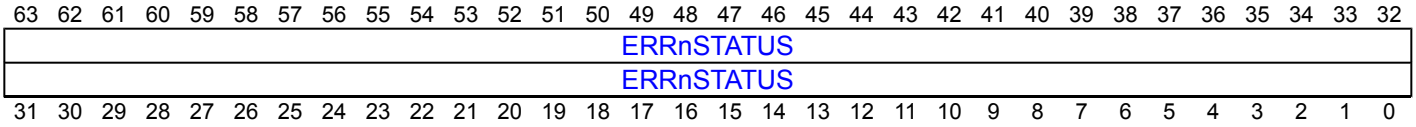
AArch64 System register ERXSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXSTATUS[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXSTATUS_EL1 are UNDEFINED.

Attributes

ERXSTATUS_EL1 is a 64-bit register.

Field descriptions



ERRnSTATUS, bits [63:0]

ERXSTATUS_EL1 accesses ERR<n>STATUS, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXSTATUS_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXSTATUS_EL1 is RAZ/WI.
- Direct reads and writes of ERXSTATUS_EL1 are NOPs.
- Direct reads and writes of ERXSTATUS_EL1 are UNDEFINED.

ERR<n>STATUS describes additional constraints that also apply when ERR<n>STATUS is accessed through ERXSTATUS_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXSTATUS_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b010

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ERXSTATUS_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXSTATUS_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXSTATUS_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXSTATUS_EL1;
```

```
MSR ERXSTATUS_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b010

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ERXSTATUS_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXSTATUS_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXSTATUS_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXSTATUS_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ESR_EL1, Exception Syndrome Register (EL1)

The ESR_EL1 characteristics are:

Purpose

Holds syndrome information for an exception taken to EL1.

Configuration

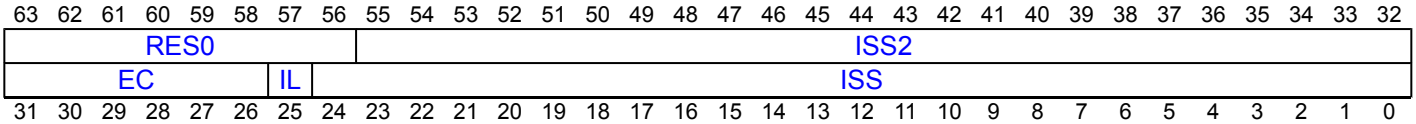
AArch64 System register ESR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DFSR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ESR_EL1 are UNDEFINED.

Attributes

ESR_EL1 is a 64-bit register.

Field descriptions



ESR_EL1 is made UNKNOWN as a result of an exception return from EL1.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL1, the value of ESR_EL1 is UNKNOWN. The value written to ESR_EL1 must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

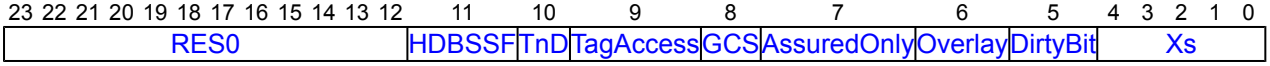
Bits [63:56]

Reserved, RES0.

ISS2, bits [55:32]

ISS2 encoding for an exception, the bit assignments are:

ISS2 encoding for an exception from a Data Abort



Bits [23:12]

Reserved, RES0.

HDBSSF, bit [11]

When FEAT_HDBSS is implemented, FEAT_NV is implemented, IsSecondStage(Fault), and DFSC == 0b0011xx || DFSC == 0b01001x || DFSC == 0b0101xx || DFSC == 0b10001x || DFSC == 0b1001xx:

Indicates that the fault was caused by the HDBSS.

When DFSC indicates a Permission fault, this field indicates whether that fault was caused by the HDBSS being full.

When DFSC indicates a synchronous External abort on translation table walk or hardware update of translation table, this field indicates whether the fault was caused by a write to the HDBSS.

When DFSC indicates a Granule Protection Fault on translation table walk or hardware update of translation table, this field indicates whether the fault was caused by a write to the HDBSS.

HDBSSF	Meaning
0b0	Fault was not caused by HDBSS.
0b1	Fault was caused by HDBSS.

This field only applies for stage 2 Permission faults.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TnD, bit [10]
When FEAT_MTE_CANONICAL_TAGS is implemented:

Tag not Data.

If a memory access generates a Data Abort for a stage 1 Permission fault, this field indicates whether the fault is due to an Allocation Tag access.

TnD	Meaning
0b0	Permission fault is not due to a write of an Allocation Tag to Canonically Tagged memory.
0b1	Permission fault is due to a write of an Allocation Tag to Canonically Tagged memory.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TagAccess, bit [9]
When FEAT_MTE_PERM is implemented and FEAT_NV is implemented:

NoTagAccess fault.

When EL2 provides information to EL1 regarding a Stage 2 Data Abort, this field indicates whether the fault is due to the NoTagAccess memory attribute.

TagAccess	Meaning
0b0	Permission fault is not due to the NoTagAccess memory attribute.
0b1	Permission fault is due to the NoTagAccess memory attribute.

For all other Data Aborts this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

GCS, bit [8]
When FEAT_GCS is implemented:

Guarded Control Stack data access.

If a memory access generates a Data Abort, this field indicates whether the fault is due to a Guarded Control Stack data access.

GCS	Meaning
0b0	The Data Abort is not due to a Guarded control stack data access.
0b1	The Data Abort is due to a Guarded control stack data access. The ISV field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]
When FEAT_THE is implemented and FEAT_NV is implemented:

AssuredOnly flag.

If EL2 provides information regarding a stage 2 Data Abort to EL1, then this field holds information about the fault.

If this field is 1 and ESR_EL1.GCS is also 1 then the AssuredOnly check might have been the result of VTCR_EL2.GCSH configuration.

AssuredOnly	Meaning
0b0	The Data Abort is not due to AssuredOnly.
0b1	The Data Abort is due to AssuredOnly.

For all other Data Aborts this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Overlay, bit [6]
When FEAT_S1POE is implemented:

Overlay flag.

If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]
When FEAT_S1PIE is implemented:

DirtyBit flag.

If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

For any other fault or Access, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Xs, bits [4:0]
When FEAT_LS64 is implemented:

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort exception for a Translation fault, Access flag fault, or Permission fault, then this field holds register specifier, Xs.

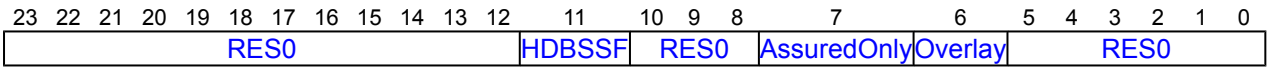
When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort exception for a Translation fault, Access flag fault, or Permission fault, then this field holds register specifier, Xs.

Otherwise, this field is RES0.

Otherwise:

Reserved, RES0.

ISS2 encoding for an exception from an Instruction Abort



Bits [23:12]

Reserved, RES0.

HDBSSF, bit [11]

When FEAT_HDBSS is implemented and FEAT_NV is implemented:

Indicates that the fault was caused by the HDBSS.

When IFSC indicates a Permission fault, this field indicates whether that fault was caused by the HDBSS being full.

When IFSC indicates a synchronous External abort on translation table walk or hardware update of translation table, this field indicates whether the fault was caused by a write to the HDBSS.

When IFSC indicates a Granule Protection Fault on translation table walk or hardware update of translation table, this field indicates whether the fault was caused by a write to the HDBSS.

HDBSSF	Meaning
0b0	Fault was not caused by HDBSS.
0b1	Fault was caused by HDBSS.

This field only applies for stage 2 Permission faults.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [10:8]

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented and FEAT_NV is implemented:

AssuredOnly flag.

If EL2 provides information regarding a stage 2 Instruction Abort to EL1, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	The Instruction Abort is not due to AssuredOnly.
0b1	The Instruction Abort is due to AssuredOnly.

For all other Instruction Aborts this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Overlay, bit [6]
When FEAT_S1POE is implemented:

Overlay flag.

If a memory access generates a Instruction Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Instruction Abort is not due to Overlay Permissions.
0b1	Instruction Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

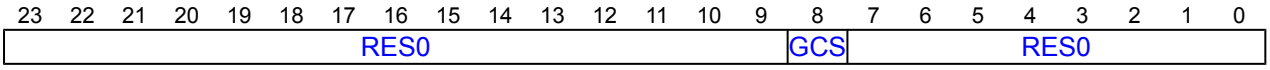
Otherwise:

Reserved, RES0.

Bits [5:0]

Reserved, RES0.

ISS2 encoding for an exception from a Watchpoint exception



Bits [23:9]

Reserved, RES0.

GCS, bit [8]
When FEAT_GCS is implemented:

Guarded control stack data access.

Indicates that the Watchpoint exception is due to a Guarded control stack data access.

GCS	Meaning
0b0	The Watchpoint exception is not due to a Guarded control stack data access.
0b1	The Watchpoint exception is due to a Guarded control stack data access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

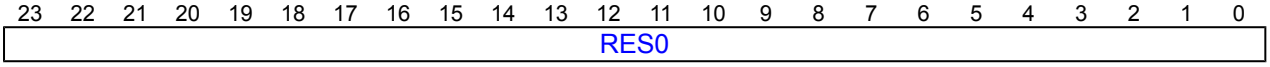
Otherwise:

Reserved, RES0.

Bits [7:0]

Reserved, RES0.

ISS2 encoding for all other exceptions



Bits [23:0]

Reserved, RES0.

EC, bits [31:26]

Exception Class. Indicates the reason for the exception that this register holds information about.

For each EC value, the table references a subsection that gives information about:

- The cause of the exception, for example the configuration required to enable the trap.
- The encoding of the associated ISS.

Possible values of the EC field are:

EC	Meaning	ISS	ISS2	Applies when
0b000000	Unknown reason.	ISS encoding for exceptions with an unknown reason	ISS2 encoding for all other exceptions	
0b000001	Trapped WF* instruction execution. Conditional WF* instructions that fail their condition code check do not cause an exception.	ISS encoding for an exception from a WF* instruction	ISS2 encoding for all other exceptions	
0b000011	Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC value 0b000000.	ISS encoding for an exception from an MCR or MRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000100	Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC value 0b000000.	ISS encoding for an exception from an MCRR or MRRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000101	Trapped MCR or MRC access with (coproc==0b1110).	ISS encoding for an exception from an MCR or MRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000110	Trapped LDC or STC access. The only architected uses of these instruction are: • An STC to write data to memory from DBGDTRRXint. • An LDC to read data from memory to DBGDTRTXint.	ISS encoding for an exception from an LDC or STC instruction	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000111	Access to SME, SVE, Advanced SIMD or floating-point functionality trapped by CPACR_EL1.FPEN, CPTR_EL2.FPEN, CPTR_EL2.TFP, or CPTR_EL3.TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2.TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000.	ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps	ISS2 encoding for all other exceptions	
0b001001	Trapped use of a Pointer authentication instruction.	ISS encoding for an exception from a trapped Pointer Authentication instruction	ISS2 encoding for all other exceptions	When FEAT_PAuth is implemented
0b001010	Trapped execution of any instruction not covered by other EC values.	ISS encoding for an exception from any other instruction	ISS2 encoding for all other exceptions	When FEAT_LS64 is implemented
0b001100	Trapped MRRC access with (coproc==0b1110).	ISS encoding for an exception from an MCRR or MRRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented

0b001101	Branch Target Exception.	ISS encoding for an exception from Branch Target Identification instruction	ISS2 encoding for all other exceptions	When FEAT_BTI is implemented
0b001110	Illegal Execution state.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault	ISS2 encoding for all other exceptions	
0b010001	SVC instruction execution in AArch32 state.	ISS encoding for an exception from HVC or SVC instruction execution	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b010100	Trapped MSRR, MRRS or System instruction execution in AArch64 state, that is not reported using EC value 0b000000.	ISS encoding for an exception from MSRR, MRRS, or 128-bit System instruction execution in AArch64 state	ISS2 encoding for all other exceptions	When FEAT_SYSREG128 is implemented or FEAT_SYSINSTR128 is implemented
0b010101	SVC instruction execution in AArch64 state.	ISS encoding for an exception from HVC or SVC instruction execution	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b011000	Trapped MSR, MRS or System instruction execution in AArch64 state, that is not reported using EC values 0b000000, 0b000001, or 0b000111. This includes all instructions that cause exceptions that are part of the encoding space defined in 'System instruction class encoding overview', except for those exceptions reported using EC values 0b000000, 0b000001, or 0b000111.	ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b011001	Access to SVE functionality trapped as a result of CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ, that is not reported using EC value 0b000000.	ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ	ISS2 encoding for all other exceptions	When FEAT_SVE is implemented
0b011010	Trapped ERET, ERETAA, or ERETAB instruction execution.	ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction	ISS2 encoding for all other exceptions	When FEAT_FGT is implemented or FEAT_NV is implemented
0b011011	Exception from an access to a TSTART instruction at EL0 when SCTLR_EL1.TME0 == 0, EL0 when SCTLR_EL2.TME0 == 0, at EL1 when SCTLR_EL1.TME == 0, at EL2 when	ISS encoding for an exception from a TSTART instruction	ISS2 encoding for all other exceptions	When FEAT_TME is implemented

	SCTLR_EL2.TME == 0 or at EL3 when SCTLR_EL3.TME == 0.			
0b011100	Exception from a PAC Fail	ISS encoding for a PAC Fail exception	ISS2 encoding for all other exceptions	When FEAT_FPAC is implemented
0b011101	Access to SME functionality trapped as a result of CPACR_EL1.SMEN, CPTR_EL2.SMEN, CPTR_EL2.TSM, CPTR_EL3.ESM, or an attempted execution of an instruction that is illegal because of the value of PSTATE.SM or PSTATE.ZA, that is not reported using EC value 0b000000.	ISS encoding for an exception due to SME functionality	ISS2 encoding for all other exceptions	When FEAT_SME is implemented
0b100000	Instruction Abort from a lower Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.	ISS encoding for an exception from an Instruction Abort	ISS2 encoding for an exception from an Instruction Abort	
0b100001	Instruction Abort taken without a change in Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.	ISS encoding for an exception from an Instruction Abort	ISS2 encoding for an exception from an Instruction Abort	
0b100010	PC alignment fault exception.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault	ISS2 encoding for all other exceptions	
0b100100	Data Abort exception from a lower Exception level. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.	ISS encoding for an exception from a Data Abort	ISS2 encoding for an exception from a Data Abort	
0b100101	Data Abort exception taken without a change in Exception level. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External	ISS encoding for an exception from a Data Abort	ISS2 encoding for an exception from a Data Abort	

	aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.			
0b100110	SP alignment fault exception.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault	ISS2 encoding for all other exceptions	
0b100111	Memory Operation Exception.	ISS encoding for an exception from the Memory Copy and Memory Set instructions	ISS2 encoding for all other exceptions	When FEAT_MOPS is implemented
0b101000	Trapped floating-point exception taken from AArch32 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.	ISS encoding for an exception from a trapped floating-point exception	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b101100	Trapped floating-point exception taken from AArch64 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.	ISS encoding for an exception from a trapped floating-point exception	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b101101	GCS exception.	ISS encoding for a GCS exception	ISS2 encoding for all other exceptions	When FEAT_GCS is implemented
0b101111	SError exception.	ISS encoding for an SError exception	ISS2 encoding for all other exceptions	
0b110000	Breakpoint exception from a lower Exception level.	ISS encoding for an exception from a Breakpoint or Vector Catch debug exception	ISS2 encoding for all other exceptions	
0b110001	Breakpoint exception taken without a change in Exception level.	ISS encoding for an exception from a Breakpoint or Vector Catch debug exception	ISS2 encoding for all other exceptions	
0b110010	Software Step exception from a lower Exception level.	ISS encoding for an exception from a Software Step exception	ISS2 encoding for all other exceptions	

0b110011	Software Step exception taken without a change in Exception level.	ISS encoding for an exception from a Software Step exception	ISS2 encoding for all other exceptions	
0b110100	Watchpoint exception from a lower Exception level.	ISS encoding for an exception from a Watchpoint exception	ISS2 encoding for an exception from a Watchpoint exception	
0b110101	Watchpoint exception taken without a change in Exception level.	ISS encoding for an exception from a Watchpoint exception	ISS2 encoding for an exception from a Watchpoint exception	
0b111000	BKPT instruction execution in AArch32 state.	ISS encoding for an exception from execution of a Breakpoint instruction	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b111100	BRK instruction execution in AArch64 state.	ISS encoding for an exception from execution of a Breakpoint instruction	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b111101	Profiling exception	ISS encoding for a Profiling exception	ISS2 encoding for all other exceptions	When FEAT_EBEP is implemented, or FEAT_SPE_EXC is implemented, or FEAT_TRBE_EXC is implemented

All other EC values are reserved by Arm, and:

- Unused values in the range 0b000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
- Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [25]

Instruction Length for synchronous exceptions. Possible values of this bit are:

IL	Meaning
0b0	16-bit instruction trapped.
0b1	32-bit instruction trapped. This value is also used when the exception is one of the following: • An SError exception. • An Instruction Abort exception. • A PC alignment fault exception. • An SP alignment fault exception. • A Data Abort exception for which the value of the ISV bit is 0. • An Illegal Execution state exception. • Any debug exception except for Breakpoint instruction exceptions. For Breakpoint instruction exceptions, this bit has its standard meaning: ◦ 0b0: 16-bit T32 BKPT instruction. ◦ 0b1: 32-bit A32 BKPT instruction or A64 BRK instruction. • An exception reported using EC value 0b000000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [24:0]

Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

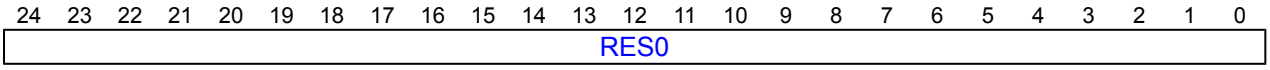
Typically, an ISS encoding has a number of subfields. When an ISS subfield holds a register number, the value returned in that field is the AArch64 view of the register number.

For an exception taken from AArch32 state, see 'Mapping of the general-purpose registers between the Execution states'.

If the AArch32 register descriptor is `0b1111`, then:

- If the instruction that generated the exception was not UNPREDICTABLE, the field takes the value `0b11111`.
- If the instruction that generated the exception was UNPREDICTABLE, the field takes an UNKNOWN value that must be either:
  - The AArch64 view of the register number of a register that might have been used at the Exception level from which the exception was taken.
  - The value `0b11111`.

ISS encoding for exceptions with an unknown reason



Bits [24:0]

Reserved, RES0.

Additional information for the ISS encoding for exceptions with an unknown reason

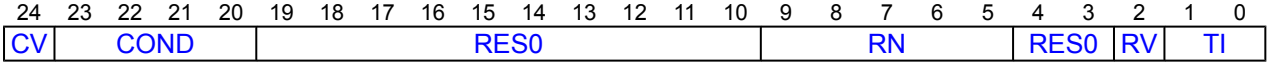
When an exception is reported using this EC value, the IL field is set to 1.

This EC value is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or that is not accessible at the current Exception level and Security state, including:
  - A read access using a System register pattern that is not allocated for reads or that does not permit reads at the current Exception level and Security state.
  - A write access using a System register pattern that is not allocated for writes or that does not permit writes at the current Exception level and Security state.
  - Instruction encodings that are unallocated.
  - Instruction encodings for instructions or System registers that are not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
- In AArch32 state, attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the SCTLR_EL1.{ITD, SED, CP15BEN} control bits.
- Attempted execution of:
  - An HVC instruction when disabled by HCR_EL2.HCD or SCR_EL3.HCE.
  - An SMC instruction when disabled by SCR_EL3.SMD.
  - An HLT instruction when disabled by EDSCR.HDE.
- Attempted execution of an MSR or MRS instruction to access SP_EL0 when the value of SPSel.SP is 0.
- Attempted execution of an MSR or MRS instruction using a _EL12 register name when the Effective value of HCR_EL2.E2H is not 1.
- Attempted execution, in Debug state, of:
  - A DCPS1 instruction when the value of HCR_EL2.TGE is 1 and EL2 is disabled or not implemented in the current Security state.
  - A DCPS2 instruction from EL1 or EL0 when EL2 is disabled or not implemented in the current Security state.
  - A DCPS3 instruction when the value of EDSCR.SDD is 1, or when EL3 is not implemented.
- When EL3 is using AArch64, attempted execution from Secure EL1 of an SRS instruction using R13_mon.

- In Debug state when the value of EDSCR.SDD is 1, the attempted execution at EL2, EL1, and EL0 of an instruction that is configured to trap to EL3.
- In AArch32 state, the attempted execution of an MRS (banked register) or an MSR (banked register) instruction to SPSR_mon, SP_mon, or LR_mon.
- An exception that is taken to EL2 because the value of HCR_EL2.TGE is 1. If the value of HCR_EL2.TGE is 0, this exception is reported using an ESR_EL1.EC value of `0b000111`.
- In Non-transactional state, attempted execution of a TCOMMIT instruction.

ISS encoding for an exception from a WF* instruction



CV, bit [24]

Condition code valid.

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to `0b1110`.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:10]

Reserved, RES0.

RN, bits [9:5]

When FEAT_WFxT is implemented:

Register Number. Indicates the register number supplied for a `WFET` or `WFIT` instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

RV, bit [2]

When FEAT_WFxT is implemented:

Register field Valid.

If TI[1] == 1, then this field indicates whether RN holds a valid register number for the register argument to the trapped `WFET` or `WFIT` instruction.

RV	Meaning
`0b0`	Register field invalid.
`0b1`	Register field valid.

If TI[1] == 0, then this field is RES0.

This field is set to 1 on a trap on `WFET` or `WFIT`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TI, bits [1:0]

Trapped instruction. Possible values of this bit are:

TI	Meaning	Applies when
`0b00`	WFI trapped.	
`0b01`	WFE trapped.	
`0b10`	WFIT trapped.	When FEAT_WFxT is implemented
`0b11`	WFET trapped.	When FEAT_WFxT is implemented

When FEAT_WFxT is implemented, this is a two bit field as shown. Otherwise, bit[1] is RES0.

The reset behavior of this field is:

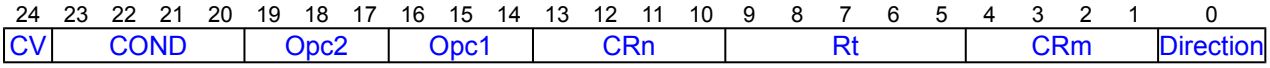
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a WF* instruction

The following fields describe configuration settings for generating this exception:

- HCR.{TWE, TWI}.
- SCTLR_EL1.{nTWE, nTWI}.
- SCTLR_EL2.{nTWE, nTWI}.
- HCR_EL2.{TWE, TWI}.
- SCR_EL3.{TWE, TWI}.

ISS encoding for an exception from an MCR or MRC access



CV, bit [24]

Condition code valid.

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc2, bits [19:17]

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value 0b000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [16:14]

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value `0b111`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

If the Rt value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
  - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
  - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value `0b0000`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
`0b0`	Write to System register space. MCR instruction.
`0b1`	Read from System register space. MRC or VMRS instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an MCR or MRC access

The following fields describe configuration settings for generating exceptions from an MCR or MRC access using coproc `0b1111`, that are reported using EC value `0b000011`:

- If FEAT_TIDCP1 is implemented, SCTLR_EL1.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch32 state, trapped to EL1.
- CNTKCTL_EL1.{EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- PMUSERENR_EL0.{ER, CR, SW, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- AMUSERENR_EL0.EN, for accesses to Activity Monitors registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- HCR.{TRVM, TVM} and HCR_EL2.{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, trapped to EL2.
- HCR.TTLB and HCR_EL2.TTLB, for execution of TLB maintenance instructions at EL1 using AArch32 state, trapped to EL2.
- HCR.{TSW, TPC, TPU} and HCR_EL2.{TSW, TPC, TPU} for execution of cache maintenance instructions at EL0 and EL1 using AArch32 state, trapped to EL2.
- HCR.TAC and HCR_EL2.TACR, for accesses to the Auxiliary Control Register at EL1 using AArch32 state, trapped to EL2.
- HCR.TIDCP and HCR_EL2.TIDCP, for accesses to lockdown, DMA, and TCM operations at EL0 and EL1 using AArch32 state, trapped to EL2.
- If FEAT_TIDCP1 is implemented, SCTLR_EL2.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch32 state, trapped to EL2.
- HCR.{TID1, TID2, TID3} and HCR_EL2.{TID1, TID2, TID3}, for accesses to ID registers at EL0 and EL1 using AArch32 state, trapped to EL2.
- HCR2.TERR, for Non-secure accesses to error record registers at EL1 using AArch32 state, trapped to EL2.
- HCPTR.TCPAC and CPTR_EL2.TCPAC, for accesses to CPACR_EL1 or CPACR using AArch32 state, trapped to EL2.
- HSTR.T<n> and HSTR_EL2.T<n>, for accesses to System registers using AArch32 state, trapped to EL2.
- CNTHCTL.PL1PCEN and CNTHCTL_EL2.EL1PCEN, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HDCR.TTRF, for Non-secure accesses to trace filter control registers from system registers using AArch32 state, trapped to EL2.
- HDCR.{TPM, TPMCR} and MDCR_EL2.{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HCPTR.TAM and CPTR_EL2.TAM, for accesses to Activity Monitors registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- CPTR_EL3.TCPAC, for accesses to CPACR from EL1 and EL2, and accesses to HCPTR from EL2 using AArch32 state, trapped to EL3.
- MDCR_EL3.TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- CPTR_EL3.TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- If FEAT_FGT is implemented, access to some registers at EL0, trapped to EL2.

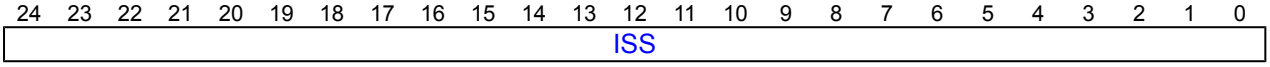
The following fields describe configuration settings for generating exceptions from an MCR or MRC access using coproc `0b1110`, that are reported using EC value `0b000101`:

- CPACR_EL1.TTA for accesses to trace registers, trapped to EL1 or EL2.
- MDSCR_EL1.TDCC, for accesses to the Debug Communications Channel (DCC) registers at EL0 and EL1 using AArch32 state, trapped to EL1 or EL2.
- If FEAT_FGT is implemented, MDCR_EL2.TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and MDCR_EL3.TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- HCR.TID0 and HCR_EL2.TID0, for accesses to the JIDR register in the ID group 0 at EL0 and EL1 using AArch32, trapped to EL2.
- HCPTR.TTA and CPTR_EL2.TTA, for accesses to trace registers using AArch32, trapped to EL2.
- HDCR.TDRA and MDCR_EL2.TDRA, for accesses to Debug ROM registers DBGDRAR and DBGDSAR using AArch32, trapped to EL2.
- HDCR.TDOSA and MDCR_EL2.TDOSA, for accesses to powerdown debug registers, using AArch32 state, trapped to EL2.
- HDCR.TDA and MDCR_EL2.TDA, for accesses to other debug registers, using AArch32 state, trapped to EL2.
- CPTR_EL3.TTA, for accesses to trace registers using AArch32, trapped to EL3.
- MDCR_EL3.TDOSA, for accesses to powerdown debug registers using AArch32, trapped to EL3.
- MDCR_EL3.TDA, for accesses to other debug registers, using AArch32, trapped to EL3.

The following fields describe configuration settings for generating exceptions from a VMSR or VMRS access, that are reported using EC value `0b001000`:

- HCR.TID0 and HCR_EL2.TID0, for accesses to the FPSID register in ID group 0 at EL1 using AArch32 state, VMRS access trapped to EL2.
- HCR.TID3 and HCR_EL2.TID3, for accesses to registers in ID group 3 including MVFR0, MVFR1 and MVFR2, VMRS access trapped to EL2.
- HCPTR.{TCP10, TCP11}, for Non-secure accesses to FPSCR, FPSID, FPEXC, MVFR0, MVFR1, and MVFR2, trapped to EL2.

ISS encoding for an exception from any other instruction

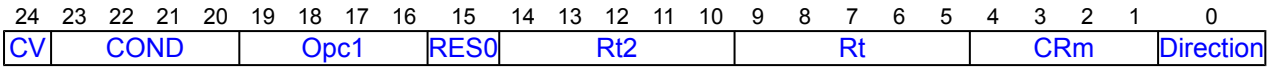


ISS, bits [24:0]

ISS	Meaning	Applies when
0b00000000000000000000000000	ST64BV instruction trapped.	When FEAT_LS64_V is implemented
0b00000000000000000000000001	ST64BV0 instruction trapped.	When FEAT_LS64_ACCDATA is implemented
0b00000000000000000000000010	LD64B or ST64B instruction trapped.	When FEAT_LS64 is implemented

All other values are reserved.

ISS encoding for an exception from an MCRR or MRRC access



CV, bit [24]

Condition code valid.

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.

        - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
    - For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [19:16]

The Opc1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

Rt2, bits [14:10]

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer.

If the Rt2 value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt2 value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
    - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
    - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer.

If the Rt value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
    - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
    - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write to System register space. MCRR instruction.
0b1	Read from System register space. MRRC instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an MCRR or MRRC access

The following fields describe configuration settings for generating exceptions from an MCRR or MRRC access using coproc 0b1111, that are reported using EC value 0b000100:

- CNTKCTL_EL1.{EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- PMUSERENR_EL0.{CR, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- AMUSERENR_EL0.{EN}, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 using AArch32 state, trapped to EL1 or EL2.
- HCR.{TRVM, TVM} and HCR_EL2.{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, trapped to EL2.
- HCR2.TERR, for Non-secure accesses to error record registers at EL1 using AArch32 state, trapped to EL2.
- HSTR.T<n> and HSTR_EL2.T<n>, for accesses to System registers using AArch32 state, trapped to EL2.
- CNTHCTL.{PL1PCEN, PL1PCTEN} and CNTHCTL_EL2.{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HDCR.TPM and MDCR_EL2.{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HCPTR.TAM and CPTR_EL2.TAM, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 and EL1 using AArch32 state, trapped to EL2.
- MDCR_EL3.TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- CPTR_EL3.TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- If FEAT_FGT is implemented, HDFGRTR_EL2.PMCCNTR_EL0 for MRRC access and HDFGWTR_EL2.PMCCNTR_EL0 for MCRR access to PMCCNTR at EL0, trapped to EL2.

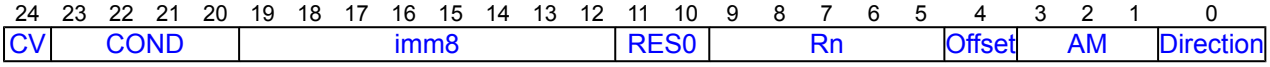
The following fields describe configuration settings for generating exceptions from an MCRR or MRRC access using coproc 0b1110, that are reported using EC value 0b001100:

- MDSCR_EL1.TDCC, for accesses to the Debug ROM registers DBGDSAR and DBGDRAR at EL0 using AArch32 state, trapped to EL1 or EL2.
- HDCR.TDRA and MDCR_EL2.TDRA, for accesses to Debug ROM registers DBGDRAR and DBGDSAR using AArch32, trapped to EL2.
- MDCR_EL3.TDA, for accesses to debug registers, using AArch32, trapped to EL3.
- CPACR_EL1.TTA for accesses to trace registers using AArch32, trapped to EL1 or EL2.
- HCPTR.TTA and CPTR_EL2.TTA, for accesses to trace registers using AArch32, trapped to EL2.
- CPTR_EL3.TTA, for accesses to trace registers using AArch32, trapped to EL3.

Note

If the Armv8-A architecture is implemented with an ETMv4 implementation, MCRR and MRRC accesses to trace registers are UNDEFINED and the resulting exception is higher priority than an exception due to these traps.

ISS encoding for an exception from an LDC or STC instruction



CV, bit [24]

Condition code valid.

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

imm8, bits [19:12]

The immediate value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

Rn, bits [9:5]

The Rn value from the issued instruction, the general-purpose register used for the transfer.

If the Rn value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rn value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
  - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
  - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

This field is valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction. When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Offset, bit [4]

Indicates whether the offset is added or subtracted:

Offset	Meaning
`0b0`	Subtract offset.
`0b1`	Add offset.

This bit corresponds to the U bit in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AM, bits [3:1]

Addressing mode. The permitted values of this field are:

AM	Meaning
`0b000`	Immediate unindexed.
`0b001`	Immediate post-indexed.
`0b010`	Immediate offset.
`0b011`	Immediate pre-indexed.
`0b100`	For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved.
`0b110`	For a trapped STC instruction, this encoding is reserved.

The values `0b101` and `0b111` are reserved. The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in System and memory-mapped registers and translation table entries'.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write to memory. STC instruction.
0b1	Read from memory. LDC instruction.

The reset behavior of this field is:

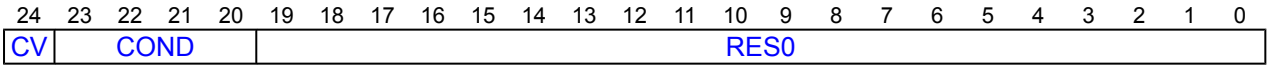
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an LDC or STC instruction

The following fields describe the configuration settings from an LDC or STC access for the traps that are reported using EC value 0b000110:

- MDSCR_EL1.TDCC, for accesses to DBGDTRTXint and DBGDTRRXint, using AArch32 state, trapped to EL1 or EL2.
- HDCR.TDA and MDCR_EL2.TDA, for accesses to DBGDTRTXint and DBGDTRRXint, using AArch32 state, trapped to EL2.
- MDCR_EL3.TDA, for accesses to DBGDTRTXint and DBGDTRRXint, using AArch32 state, trapped to EL3.
- If FEAT_FGT is implemented, MDCR_EL2.TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and MDCR_EL3.TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.

ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps



The accesses covered by this trap include:

- Execution of SVE or Advanced SIMD and floating-point instructions.
- Accesses to the Advanced SIMD and floating-point System registers.
- Execution of SME instructions.

For an implementation that does not include either SVE or support for Advanced SIMD and floating-point, the exception is reported using the EC value 0b000000.

CV, bit [24]

Condition code valid.

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:0]

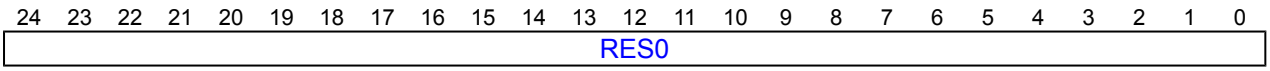
Reserved, RES0.

Additional information for the ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps

The following fields describe the configuration settings for the traps that are reported using EC value `0b000111`:

- HCPTR.{TCP10, TCP11}, for Non-secure accesses to Advanced SIMD and floating-point registers and instructions, trapped to EL2.
- HCPTR.TASE, for Non-secure accesses to Advanced SIMD functionality, trapped to EL2.
- CPACR_EL1.FPEN, for accesses to SIMD and floating-point registers trapped to EL1.
- CPTR_EL2.FPEN and CPTR_EL2.TFP, for accesses to SIMD and floating-point registers trapped to EL2.
- CPTR_EL3.TFP, for accesses to SIMD and floating-point registers trapped to EL3.

ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ



The accesses covered by this trap include:

- Execution of SVE instructions when the PE is not in Streaming SVE mode.
- Accesses to the SVE System registers, ZCR_ELx.

For an implementation that does not include SVE, the exception is reported using the EC value `0b000000`.

Bits [24:0]

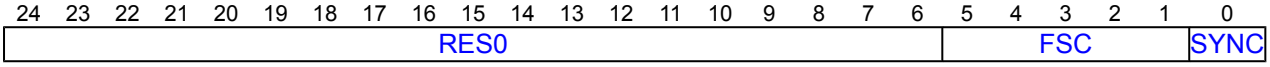
Reserved, RES0.

Additional information for the ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ

The following fields describe the configuration settings for the traps that are reported using EC value `0b011001`:

- CPACR_EL1.ZEN, for execution of SVE instructions and accesses to SVE registers at EL0 or EL1, trapped to EL1.
- CPTR_EL2.ZEN and CPTR_EL2.TZ, for execution of SVE instructions and accesses to SVE registers at EL0, EL1, or EL2, trapped to EL2.
- CPTR_EL3.EZ, for execution of SVE instructions and accesses to SVE registers from all Exception levels, trapped to EL3.

ISS encoding for a Profiling exception



Bits [24:6]

Reserved, RES0.

FSC, bits [5:1]

Indicates why the Profiling exception was generated.

FSC	Meaning	Applies when
0b00000	PMU Profiling exception. One of the following applies, and ESR_EL1.SYNC describes which: • The exception was generated because at least one PMU counter overflow status flag was 1, and was taken asynchronously. • The exception was generated because FEAT_SEBEP is implemented and PSTATE.PPEND was 1, and was taken synchronously.	When FEAT_EBEP is implemented
0b00001	Profiling Buffer management event. The exception was generated because PMBSR_EL1.S was 1.	When FEAT_SPE_EXC is implemented
0b00010	Trace buffer management event. The exception was generated because TRBSR_EL1.IRQ was 1.	When FEAT_TRBE_EXC is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SYNC, bit [0]

Indicates whether the Profiling exception was taken synchronously or asynchronously.

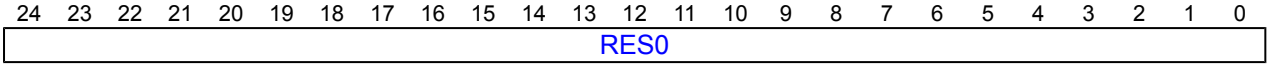
SYNC	Meaning	Applies when
0b0	The exception was taken asynchronously.	
0b1	The exception was taken synchronously.	When FEAT_SEBEP is implemented

If ESR_EL1.FSC does not indicate a PMU Profiling exception, or FEAT_SEBEP is not implemented, then the only permitted value is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault



Bits [24:0]

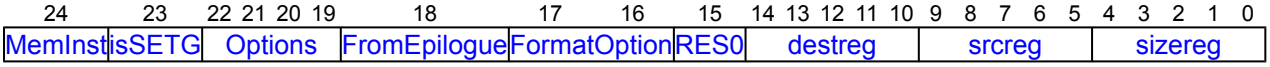
Reserved, RES0.

Additional information for the ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault

There are no configuration settings for generating Illegal Execution state exceptions and PC alignment fault exceptions. For more information about PC alignment fault exceptions, see 'PC alignment checking'.

'SP alignment checking' describes the configuration settings for generating SP alignment fault exceptions.

ISS encoding for an exception from the Memory Copy and Memory Set instructions



MemInst, bit [24]

Indicates the memory instruction class causing the exception.

MemInst	Meaning
0b0	CPYFE*, CPYFM*, CPYE*, and CPYM* instructions.
0b1	SETE*, SETM*, SETGE*, and SETGM* instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

isSETG, bit [23]

Indicates whether the instruction belongs to SETGM* or SETGE* class of instruction.

isSETG	Meaning
0b0	Not a SETGM* or SETGE* instruction.
0b1	SETGM* or SETGE* instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Options, bits [22:19]

Options : the Options field of the instruction.

For Memory Copy instructions, bits[22:19] forms the Options field, which holds the bits[15:12] of the instruction.

For Memory Set instructions:

- Bits[22:21] are RES0.
- Bits[20:19] form the Options field, which holds the bits[13:12] of the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FromEpilogue, bit [18]

Indicates whether the instruction belongs to the epilogue class of Memory Copy or Memory Set instructions.

FromEpilogue	Meaning
0b0	Not an epilogue instruction.
0b1	CPYE*, CPYFE*, SETE*, or SETGE* instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FormatOption, bits [17:16]

Reports the Option used to encode the initial Xs, Xd, and Xn register values provided to the instruction that generated the exception.

FormatOption	Meaning
0b00	Option B.
0b01	Option A.
0b10	Option A.
0b11	Option B.

For more information, see Memory Copy and Memory Set exceptions.

Note

This field was previously presented as two separate bits, WrongOption, bit[17] and OptionA, bit [16], which were already expected to be used together and not individually.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

destreg, bits [14:10]

The destination register value from the issued instruction, containing the destination address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

srcreg, bits [9:5]

The source register value from the issued instruction, containing either the source address or the source data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

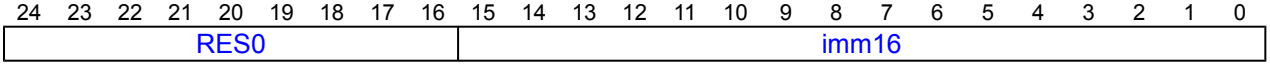
sizereg, bits [4:0]

The size register value from the issued instruction, containing the number of bytes to be transfered or set.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from HVC or SVC instruction execution



Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, and for an A64 SVC instruction, this is the value of the imm16 field of the issued instruction.

For an A32 or T32 SVC instruction:

- If the instruction is unconditional, then:
  - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
  - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from HVC or SVC instruction execution

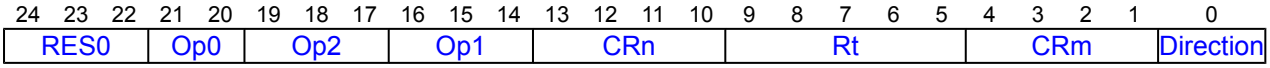
In AArch32 state, the HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.

For T32 and A32 instructions, see 'SVC' and 'HVC'.

For A64 instructions, see 'SVC' and 'HVC'.

If FEAT_FGT is implemented, HFGITR_EL2.{SVC_EL1, SVC_EL0} control fine-grained traps on SVC execution.

ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state



Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

For system instructions which require that the opcode Rt field is set to `0b11111`, but where the trapped instruction has a different value of Rt, an implementation is permitted to return the value `0b11111`, instead of the value of Rt from the trapped instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
`0b0`	Write access, including MSR instructions.
`0b1`	Read access, including MRS instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state

For exceptions caused by System instructions, see 'System instructions' subsection of 'Branches, exception generating and System instructions' for the encoding values returned by an instruction.

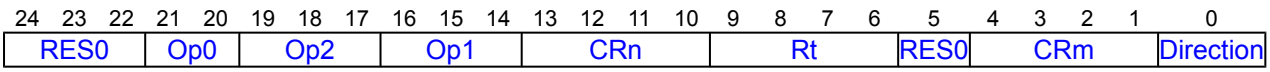
The following fields describe configuration settings for generating the exception that is reported using EC value `0b011000`:

- If FEAT_TIDCP1 is implemented, SCTLR_EL1.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch64 state, MSR or MRS access trapped to EL1.
- SCTLR_EL1.UCI, for execution of cache maintenance instructions using AArch64 state, execution is trapped to EL1 or EL2.
- SCTLR_EL1.UCT, for accesses to CTR_EL0 using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- SCTLR_EL1.DZE, for execution of DC ZVA instructions using AArch64 state, execution is trapped to EL1 or EL2.
- SCTLR_EL1.UMA, for accesses to the PSTATE interrupt masks using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- CPACR_EL1.TTA, for accesses to the trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- MDSCR_EL1.TDCC, for accesses to the Debug Communications Channel (DCC) registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- If FEAT_FGT is implemented, MDCR_EL2.TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and MDCR_EL3.TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- CNTKCTL_EL1.{EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN} accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- PMUSERENR_EL0, for accesses to the Performance Monitor registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- AMUSERENR_EL0.EN, for accesses to Activity Monitors registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.

- HCR_EL2.{TRVM, TVM}, for accesses to virtual memory control registers using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.TDZ, for execution of DC ZVA instructions using AArch64 state, execution is trapped to EL2.
- HCR_EL2.TTLB, for execution of TLB maintenance instructions using AArch64 state, execution is trapped to EL2.
- HCR_EL2.{TSW, TPC, TPU}, for execution of cache maintenance instructions using AArch64 state, execution is trapped to EL2.
- HCR_EL2.TACR, for accesses to the Auxiliary Control Register, ACTLR_EL1, using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.TIDCP, for accesses to lockdown, DMA, and TCM operations using AArch64 state, MSR or MRS access trapped to EL2.
- If FEAT_TIDCP1 is implemented, SCTLR_EL2.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.{TID1, TID2, TID3}, for accesses to ID group 1, ID group 2 or ID group 3 registers, using AArch64 state, MSR or MRS access trapped to EL2.
- If FEAT_MTE2 is implemented, HCR_EL2.TID5, for accesses to GMID_EL1, using AArch64 state, MRS or MSR access at EL1, trapped to EL2.
- If FEAT_IDTE3 is implemented, SCR_EL3.TID3, for accesses to ID group 3 registers using AArch64 state, at EL1 and EL2, trapped to EL3.
- CPTR_EL2.TCPAC, for accesses to CPACR_EL1, using AArch64 state, MSR or MRS access trapped to EL2.
- CPTR_EL2.TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.TTRF, for accesses to the trace filter control register, TRFCR_EL1, using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.TDRA, for accesses to Debug ROM registers, using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.TDOSA, for accesses to powerdown debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- CNTHCTL_EL2.{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.TDA, for accesses to debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.{TPM, TPMCR}, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL2.
- CPTR_EL2.TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.{NV, NV1}, for Nested virtualization register access, using AArch64 state, MSR or MRS access, trapped to EL2.
- HCR_EL2.AT, for execution of AT S1E* instructions, using AArch64 state, execution is trapped to EL2.
- HCR_EL2.{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access, trapped to EL2.
- SCR_EL3.APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL3.
- SCR_EL3.ST, for accesses to the Counter-timer Physical Secure timer registers, using AArch64 state, MSR or MRS access trapped to EL3.
- SCR_EL3.{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL3.
- CPTR_EL3.TCPAC, for accesses to CPTR_EL2 and CPACR_EL1 using AArch64 state, MSR or MRS access trapped to EL3.
- CPTR_EL3.TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TTRF, for accesses to the trace filter control registers, TRFCR_EL1 and TRFCR_EL2, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TDA, for accesses to debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TDOSA, for accesses to powerdown debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TPM, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL3.
- If FEAT_SPE is implemented:
  - MDCR_EL3.NSPB for accesses to Statistical Profiling and Profiling Buffer control registers, using AArch64 state, MSR or MRS access at EL1 and EL2 trapped to EL3.
  - MDCR_EL2.TPMS for accesses to SPE registers, using AArch64 state, MSR or MRS access at EL1 trapped to EL2.
- CPTR_EL3.TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access, trapped to EL3.
- If FEAT_EVT is implemented, the following registers control traps for EL1 and EL0 Cache controls that use this EC value:
  - HCR_EL2.{TTLBOS, TTLBIS, TICAB, TOCU, TID4}.
  - HCR2.{TTLBIS, TICAB, TOCU, TID4}.
- If FEAT_FGT is implemented:
  - SCR_EL3.FGTEn, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
  - HFGRTR_EL2 for reads and HFGWTR_EL2 for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 trapped to EL2.
  - HFGITR_EL2 for execution of system instructions, MSR or MRS access trapped to EL2.
  - HDFGRTR_EL2 for reads and HDFGWTR_EL2 for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 state trapped to EL2.
  - HAFGRTR_EL2 for reads of Activity Monitor counters, using AArch64 state, MRS access at EL0 and EL1 trapped to EL2.
- If FEAT_RNG_TRAP is implemented, SCR_EL3.TRNDR for reads of RNDR and RNDRRS using AArch64 state, MRS access trapped to EL3.
- If FEAT_SME is implemented:
  - CPTR_EL3.ESM, for MSR or MRS accesses to SMPRI_EL1 at EL1, EL2, and EL3, trapped to EL3.
  - CPTR_EL3.ESM, for MSR or MRS accesses to SMPRIMAP_EL2 at EL2 and EL3, trapped to EL3.
  - SCTLR_EL1.EnTP2, for MSR or MRS accesses to TPIDR2_EL0 at EL0, trapped to EL1 or EL2.
  - SCTLR_EL2.EnTP2, for MSR or MRS accesses to TPIDR2_EL0 at EL0, trapped to EL2.
  - SCR_EL3.EnTP2, for MSR or MRS accesses to TPIDR2_EL0 at EL0, EL1, and EL2, trapped to EL3.
- If FEAT_FPMR is implemented:

        - SCTLR_EL1.EnFPM, for accesses to FPMR at EL0, trapped to EL1 or EL2.
        - SCTLR_EL2.EnFPM, for accesses to FPMR at EL0, trapped to EL2.
        - HCRX_EL2.EnFPM, for accesses to FPMR at EL0 and EL1, trapped to EL2.
        - SCR_EL3.EnFPM, for accesses to FPMR at EL0, EL1, and EL2, trapped to EL3.
- If FEAT_NMI is implemented, HCRX_EL2.TALLINT, for MSR writes of ALLINT at EL1, trapped to EL2.
- If FEAT_FGT2 is implemented:
    - SCR_EL3.FGTEn2, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
    - HFGRTR2_EL2 for reads and HFGWTR2_EL2 for writes of registers, using AArch64 state, using MSR or MRS access at EL1 trapped to EL2.
    - HDFGRTR2_EL2 for reads and HDFGWTR2_EL2 for writes of registers, using AArch64 state, using MSR or MRS access at EL0 and EL1 trapped to EL2.
    - HFGITR2_EL2 for execution of system instructions, MSR or MRS access trapped to EL2.
- If FEAT_ITE is implemented, MDCR_EL3.EnITE, for accesses to Instrumentation trace registers, using AArch64 state, MSR or MRS access, trapped to EL3.
- If FEAT_MEC is implemented, SCR_EL3.MECEn, for accesses to MECID registers at EL2, trapped to EL3.
- If FEAT_SPE_FDS is implemented, MDCR_EL3.EnPMS3 for accesses to SPE registers, using AArch64 state, MSR or MRS access at EL1 and EL2 trapped to EL3.
- If FEAT_SPE_nVM is implemented, MDCR_EL3.EnPMS4 for accesses to SPE registers, using AArch64 state, MSR or MRS access at EL1 and EL2 trapped to EL3.
- If FEAT_RASv2 is implemented, SCR_EL3.TWERR, for accesses to Error Record registers, MSR access at EL1 and EL2 trapped to EL3.
- If FEAT_Debugv8p9 is implemented, MDCR_EL3.EBWE for accesses of MDSELR_EL1, using AArch64 state, MRS or MSR access at EL2 and EL1 trapped to EL3.
- If FEAT_PMUv3p9, FEAT_SPMU, FEAT_EBEP, or FEAT_PMUv3_SS is implemented, MDCR_EL3.EnPM2, for accesses to PMU registers, using AArch64 state, MSR or MRS access at EL2, EL1, and EL0, trapped to EL3.
- If FEAT_PMUv3_SS is implemented, MDCR_EL3.EnPMSS, for accesses to PMU Snapshot registers, using AArch64 state, MSR or MRS access at EL2 and EL1 trapped to EL3.
- If FEAT_THE is implemented, SCR_EL3.RCWMASKEn for accesses to RCWMASK_EL1 and RCWSMASK_EL1, using AArch64 state, MSR or MRS access at EL2 and EL1 trapped to EL3.
- If FEAT_AIE is implemented, SCR_EL3.AIEn for accesses to Extended Memory Attribute registers, MSR or MRS access at EL2 and EL1 trapped to EL3.
- If FEAT_S1PIE, FEAT_S2PIE, FEAT_S1POE, or FEAT_S2POE is implemented, SCR_EL3.PIEn for accesses to Permission Indirection, Overlay registers, MSR or MRS access at EL2, EL1 and EL0 trapped to EL3.
- If FEAT_MPAM_PE_BW_CTRL is implemented, MPAMBW2_EL2.{nTRAP_MPAMBWIDR_EL1, nTRAP_MPAMBW0_EL1, nTRAP_MPAMBW1_EL1} for accesses to MPAMBW*_EL1 registers, using AArch64 state, MRS or MSR access at EL1, trapped to EL2.
- If FEAT_MPAM_PE_BW_CTRL and FEAT_SME are implemented, MPAMBW2_EL2.nTRAP_MPAMBWSM_EL1, for accesses to MPAMBWSM_EL1, using AArch64 state, MRS or MSR access at EL1, trapped to EL2.
- If FEAT_MPAM_PE_BW_CTRL is implemented, MPAMBW3_EL3.nTRAPLOWER, for accesses to MPAMBW_EL2 registers and MPAMBW_EL1 registers, using AArch64 state, MRS or MSR access at EL1, trapped to EL3.
- If FEAT_HACDBS is implemented, SCR_EL3.HACDBSEn, for accesses to HACDBSBR_EL2 and HACDBSCONS_EL2, using AArch64 state, MRS or MSR access at EL2, trapped to EL3.
- If FEAT_HDBSS is implemented, SCR_EL3.HDBSSEn, for accesses to HDBSSBR_EL2 and HDBSSPROD_EL2, using AArch64 state, MRS or MSR access at EL2, trapped to EL3.
- If FEAT_SRMASK is implemented, SCR_EL3.SRMASKEn, for MSR or MRS accesses at EL1 or EL2 to the MASK registers using AArch64 state, trapped to EL3.

ISS encoding for an exception from MSRR, MRRS, or 128-bit System instruction execution in AArch64 state



Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:6]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

Note

This value represents register pair of X[Rt:0], X[Rt:1].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write access, MSRR instructions.
0b1	Read access, MRRS instructions.

The reset behavior of this field is:

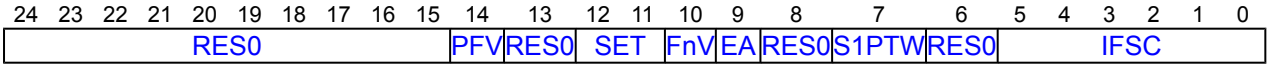
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from MSRR, MRRS, or 128-bit System instruction execution in AArch64 state

The following fields describe configuration settings for generating exceptions from an MSRR or MRRS access that are reported using EC value `0b010100`:

- If FEAT_FGT is implemented:
  - HFGRTR_EL2 for reads and HFGWTR_EL2 for writes of registers, using AArch64 state, accesses at EL1 trapped to EL2.
- If FEAT_FGT2 is implemented:
  - HFGRTR2_EL2.nRCWSMASK_EL1 for reads and HFGWTR2_EL2.nRCWSMASK_EL1 for writes of RCWSMASK_EL1, using AArch64 state, accesses at EL1 trapped to EL2.
- If FEAT_SYSREG128 is implemented:
  - SCTLR2_EL1.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL0 trapped to EL1.
  - SCTLR2_EL2.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL0 trapped to EL2.
  - HCRX_EL2.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL1 and EL0 trapped to EL2.
  - SCR_EL3.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL2, EL1, and EL0 trapped to EL3.
- If FEAT_D128 is implemented:
  - HCR_EL2.{TRVM, TVM} for accesses to TTBR0_EL1 and TTBR1_EL1, accesses at EL1 and EL0 trapped to EL2.
  - HCRX_EL2.D128En for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL1 trapped to EL2.
  - SCR_EL3.D128En for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL2 and EL1 trapped to EL3.
- If FEAT_THE is implemented, SCR_EL3.RCWMASKEn for accesses to RCWMASK_EL1 and RCWSMASK_EL1, using AArch64 state, accesses at EL2 and EL1 trapped to EL3.

ISS encoding for an exception from an Instruction Abort



When FEAT_S1POE is implemented, if a memory access generates a Instruction Abort due to a Permission fault, the ISS2 encoding for an exception from an Instruction Abort includes further information about the exception.

Bits [24:15]

Reserved, RES0.

PFV, bit [14]

When FEAT_PFAR is implemented and (IFSC == 0b010000, or IFSC IN {0b01001x}, or IFSC IN {0b0101xx}):

PFAR Valid. Describes whether the PFAR_EL1 register is valid.

PFV	Meaning
0b0	PFAR_EL1 is UNKNOWN.
0b1	PFAR_EL1 is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [13]

Reserved, RES0.

SET, bits [12:11]

When FEAT_RAS is implemented and (IFSC == 0b010000, or IFSC IN {0b01001x}, or IFSC IN {0b0101xx}):

Synchronous Error Type. Describes the PE error state after taking the Instruction Abort exception.

SET	Meaning	Applies when
0b00	Recoverable state (UER).	
0b10	Uncontainable (UC).	When FEAT_RASv2 is not implemented
0b11	Restartable state (UEO).	

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External abort exception might result in a PE state that is not recoverable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	FAR is valid.
0b1	FAR is not valid, and holds an UNKNOWN value.

This field is valid only if the IFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

IFSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	When FEAT_RAS is not implemented
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented

0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented

All other values are reserved.

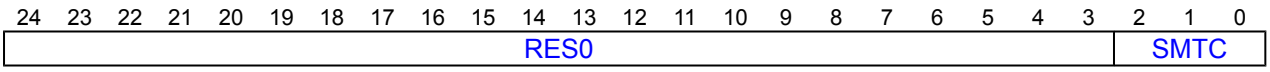
For more information about the lookup level associated with a fault, see 'The lookup level associated with MMU faults'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception due to SME functionality



The accesses covered by this trap include:

- Execution of SME instructions.
- Execution of SVE and Advanced SIMD instructions, when the PE is in Streaming SVE mode.
- Direct accesses of SVCR, SMCR_EL1, SMCR_EL2, SMCR_EL3.

Bits [24:3]

Reserved, RES0.

SMTC, bits [2:0]

SME Trap Code. Identifies the reason for instruction trapping.

SMTC	Meaning	Applies when
0b000	Access to SME functionality trapped as a result of CPACR_EL1.SMEN, CPTR_EL2.SMEN, CPTR_EL2.TSM, or CPTR_EL3.ESM, that is not reported using EC value 0b000000.	
0b001	Advanced SIMD, SVE, or SVE2 instruction trapped because PSTATE.SM is 1.	
0b010	SME instruction trapped because PSTATE.SM is 0.	
0b011	SME instruction trapped because PSTATE.ZA is 0.	
0b100	Access to the SME2 ZT0 register trapped as a result of SMCR_EL1.EZT0, SMCR_EL2.EZT0, or SMCR_EL3.EZT0.	When FEAT_SME2 is implemented

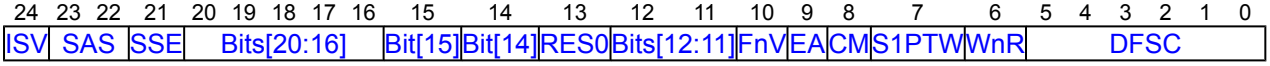
All other values are reserved.

Additional information for the ISS encoding for an exception due to SME functionality

The following fields describe the configuration settings for the traps that are reported using the EC value 0b011101:

- CPACR_EL1.SMEN, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access SVCR and SMCR_EL1 System registers at EL1 and EL0, trapped to EL1 or EL2.
- CPTR_EL2.SMEN and CPTR_EL2.TSM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access SVCR, SMCR_EL1, SMCR_EL2 at EL2, EL1, and EL0, trapped to EL2.
- CPTR_EL3.ESM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access SVCR, SMCR_EL1, SMCR_EL2, SMCR_EL3 from all Exception levels and any Security state, trapped to EL3.
- If FEAT_SME2 is implemented:
  - SMCR_EL1.EZT0, for accesses to ZT0 at EL1 and EL0, trapped to EL1 or EL2.
  - SMCR_EL2.EZT0, for accesses to ZT0 at EL2, EL1, and EL0, trapped to EL2.
  - SMCR_EL3.EZT0, for accesses to ZT0 at any Exception level, trapped to EL3.

ISS encoding for an exception from a Data Abort



The ISS2 encoding for an exception from a Data Abort includes further information about the exception when any of the following features are implemented:

- FEAT_LS64_V.
- FEAT_LS64_ACCDATA.
- FEAT_S1POE.
- FEAT_S1PIE.
- FEAT_GCS.
- FEAT_MTE_CANONICAL_TAGS.

ISV, bit [24]

Instruction Syndrome Valid. Indicates whether the syndrome information in ISS[23:14] is valid.

ISV	Meaning
0b0	No valid instruction syndrome. ISS[23:14] are RES0.
0b1	ISS[23:14] hold a valid instruction syndrome.

In ESR_EL1, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

In ESR_EL1, ISV is 1 when FEAT_LS64_V is implemented and a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

In ESR_EL1, ISV is 1 when FEAT_LS64_ACCDATA is implemented and a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For other faults reported in ESR_EL1, ISV is 0 except for the following stage 2 aborts:

- AArch64 loads and stores of a single general-purpose register (including the register specified with `0b11111`, including those with Acquire/Release semantics, but excluding Load Exclusive or Store Exclusive and excluding those with writeback).
- AArch32 instructions where the instruction:
    - Is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
    - Is not performing register writeback.
    - Is not using R15 as a source or destination register.

For these stage 2 aborts, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, and otherwise indicates whether ISS[23:14] hold a valid syndrome.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64_V is implemented and a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64_ACCDATA is implemented and a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

When FEAT_RAS is implemented, ISV is 0 for any synchronous External abort.

When FEAT_RAS is not implemented, it is IMPLEMENTATION DEFINED whether ISV is set to 1 or 0 on a synchronous External abort on a stage 2 translation table walk.

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When FEAT_MTE2 is implemented, for a synchronous Tag Check Fault abort taken to EL1, ESR_EL1.FnV is 0 and FAR_EL1 is valid.

When FEAT_MOPS is implemented, for a synchronous Data Abort on a Memory Copy and Memory Set instruction, ISV is 0.

When FEAT_MTE is implemented, for a synchronous Data Abort on an instruction that directly accesses Allocation Tags, ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SAS, bits [23:22]
When ISV == 1:

Syndrome Access Size. Indicates the size of the access attempted by the faulting operation.

SAS	Meaning
`0b00`	Byte
`0b01`	Halfword
`0b10`	Word
`0b11`	Doubleword

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is `0b11`.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is `0b11`.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is `0b11`.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSE, bit [21]
When ISV == 1:

Syndrome Sign Extend. For a byte, halfword, or word load operation, indicates whether the data item must be sign extended.

SSE	Meaning
0b0	Sign-extension not required.
0b1	Data item must be sign-extended.

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

For all other operations, this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits[20:16]
When ISV == 1:

SRT, bits [4:0] of bits [20:16]

Syndrome Register Transfer. The register number of the Wt/Xt/Rt operand of the faulting instruction.

If the exception was taken from an Exception level that is using AArch32, then this is the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states'.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ISV == 0, FEAT_RASv2 is implemented, and (DFSC == 0b010000, or DFSC IN {0b01001x}, or DFSC IN {0b0101xx}):

Bits [4:2] of bits [20:16]

Reserved, RES0.

WU, bits [1:0] of bits [20:16]

Write Update. Describes whether a store instruction that generated an External abort updated the location.

WU	Meaning
0b00	Not a store instruction or translation table update, or the location might have been updated.
0b10	Store instruction or translation table update that did not update the location.
0b11	Store instruction or translation table update that updated the location.

In the description of this field, a store instruction is any memory-writing instruction that explicitly performs a store. This includes instructions that both read and write memory.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[15]
When ISV == 1:

SF, bit [15]

Sixty Four bit general-purpose register transfer. Width of the register accessed by the instruction is 64-bit.

SF	Meaning
0b0	Instruction loads/stores a 32-bit general-purpose register.
0b1	Instruction loads/stores a 64-bit general-purpose register.

Note

This field specifies the register width identified by the instruction, not the Execution state.

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ISV == 0:

FnP, bit [15]

FAR not Precise.

FnP	Meaning	Applies when
0b0	The FAR holds the faulting virtual address that generated the Data Abort.	
0b1	The FAR holds any virtual address within the naturally-aligned granule that contains the faulting virtual address that generated a Data Abort due to an SVE contiguous vector load/store instruction, or an SME load/store instruction. For more information about the naturally-aligned fault granule, see FAR_ELx (for example, FAR_EL1).	When FEAT_SME is implemented or FEAT_SVE is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[14]
When ISV == 1:

AR, bit [14]

Acquire/Release.

AR	Meaning
0b0	Instruction did not have acquire/release semantics.
0b1	Instruction did have acquire/release semantics.

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_PFAR is implemented and (DFSC == 0b010000, or DFSC IN {0b01001x}, or DFSC IN {0b0101xx}):

PFV, bit [14]

PFAR Valid. Describes whether the PFAR_EL1 register is valid.

PFV	Meaning
0b0	PFAR_EL1 is UNKNOWN.
0b1	PFAR_EL1 is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [13]

Reserved, RES0.

Bits[12:11]
When (DFSC IN {0b00xxxx} || DFSC IN {0b10101x}) && !(DFSC IN {0b0000xx}):

LST, bits [1:0] of bits [12:11]

Load/Store Type. Used when a Translation fault, Access flag fault, or Permission fault generates a Data Abort.

LST	Meaning	Applies when
0b00	The instruction that generated the Data Abort is not specified by this field.	
0b01	An ST64BV instruction generated the Data Abort.	When FEAT_LS64_V is implemented
0b10	An LD64B or ST64B instruction generated the Data Abort.	When FEAT_LS64 is implemented
0b11	An ST64BV0 instruction generated the Data Abort.	When FEAT_LS64_ACCDATA is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_RAS is implemented and (DFSC == 0b010000, or DFSC IN {0b01001x}, or DFSC IN {0b0101xx}):

SET, bits [1:0] of bits [12:11]

Synchronous Error Type. Used when a synchronous External abort, not on a Translation table walk or hardware update of the Translation table, generated the Data Abort. Describes the PE error state after taking the Data Abort exception.

SET	Meaning	Applies when
0b00	Recoverable state (UER).	
0b10	Uncontainable (UC).	When FEAT_RASv2 is not implemented
0b11	Restartable state (UEO).	

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External abort exception might result in a PE state that is not recoverable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	FAR is valid.
0b1	FAR is not valid, and holds an UNKNOWN value.

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

CM	Meaning
0b0	The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1.
0b1	The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA, DC GVA, and DC GZVA instructions are not classified as cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

WnR	Meaning
0b0	Abort caused by an instruction reading from a memory location.
0b1	Abort caused by an instruction writing to a memory location.

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- If FEAT_RASv2 is implemented, an External abort on an Atomic access, reported with ESR_EL1.WU set to 0b00.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

DFSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Synchronous Tag Check Fault.	When FEAT_MTE2 is implemented
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	When FEAT_RAS is not implemented
0b100001	Alignment fault.	

0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented
0b110100	IMPLEMENTATION DEFINED fault (Lockdown).	
0b110101	IMPLEMENTATION DEFINED fault (Unsupported Exclusive or Atomic access).	

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The lookup level associated with MMU faults'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a trapped floating-point exception

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0	TFV	RES0												VECITR			IDF	RES0		IXF	UFF	OFF	DZF	IOF

Bit [24]

Reserved, RES0.

TFV, bit [23]

Trapped Fault Valid bit. Indicates whether the IDF, IXF, UFF, OFF, DZF, and IOF bits hold valid information about trapped floating-point exceptions.

TFV	Meaning
0b0	The IDF, IXF, UFF, OFF, DZF, and IOF bits do not hold valid information about trapped floating-point exceptions and are UNKNOWN.
0b1	One or more floating-point exceptions occurred during an operation performed while executing the reported instruction. The IDF, IXF, UFF, OFF, DZF, and IOF bits indicate trapped floating-point exceptions that occurred. For more information, see 'Floating-point exceptions and exception traps'.

It is IMPLEMENTATION DEFINED whether this field is set to 0 on an exception generated by a trapped floating-point exception from an instruction that is performing floating-point operations on more than one lane of a vector.

Note

This is not a requirement. Implementations can set this field to 1 on a trapped floating-point exception from an instruction and return valid information in the {IDF, IXF, UFF, OFF, DZF, IOF} fields.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [22:11]

Reserved, RES0.

VECITR, bits [10:8]

For a trapped floating-point exception from an instruction executed in AArch32 state this field is RES1.

For a trapped floating-point exception from an instruction executed in AArch64 state this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IDF	Meaning
0b0	Input denormal floating-point exception has not occurred.
0b1	Input denormal floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IXF	Meaning
0b0	Inexact floating-point exception has not occurred.
0b1	Inexact floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

UFF	Meaning
0b0	Underflow floating-point exception has not occurred.
0b1	Underflow floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

OFF	Meaning
0b0	Overflow floating-point exception has not occurred.
0b1	Overflow floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

DZF	Meaning
0b0	Divide by Zero floating-point exception has not occurred.
0b1	Divide by Zero floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IOF	Meaning
0b0	Invalid Operation floating-point exception has not occurred.
0b1	Invalid Operation floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

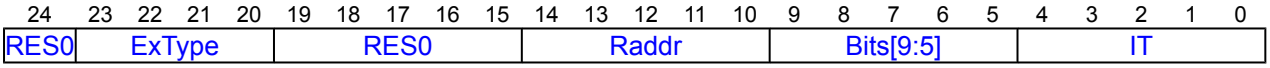
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a trapped floating-point exception

In an implementation that supports the trapping of floating-point exceptions:

- From an Exception level using AArch64, the FPCR.{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.
- From an Exception level using AArch32, the FPSCR.{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.

ISS encoding for a GCS exception



Bit [24]

Reserved, RES0.

ExType, bits [23:20]

The first level classification of GCS exceptions.

ExType	Meaning
0b0000	The exception reported is a Guarded Control Stack Data Check Exception.
0b0001	The exception reported is an EXLOCK Exception.
0b0010	The exception reported is a trap exception on GCSSTR or GCSSTTR instruction execution.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:15]

Reserved, RES0.

Raddr, bits [14:10]

When ExType == 0b0010 :

Indicates the data address register number supplied in the instruction that has been trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits[9:5]

When ExType == 0b0000 :

Rn, bits [4:0] of bits [9:5]

Indicates a register number used by the instruction that caused the Guarded Control Stack Data Check Exception.

For a procedure return instruction reported with ESR_EL1.ISS.IT as 0b00000, contains the register number for the register which contains the target address of the branch.

For a GCSPOPM instruction reported with ESR_EL1.ISS.IT as 0b00001, contains the register number for the register which is the destination register of the instruction.

For a procedure return instruction reported with ESR_EL1.ISS.IT as `0b00010` or `0b00011`, contains the value `0b11110`, indicating X30.

For a GCSSS1 instruction reported with ESR_EL1.ISS.IT as `0b00100`, contains the register number for the register which is the input register of the instruction.

If ESR_EL1.ISS.IT is reported as `0b00101` or `0b01000`, this field is UNKNOWN

If ESR_EL1.ISS.IT is reported as `0b01001`, this field is `0b11111`

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ExType == 0b0010 :

Rvalue, bits [4:0] of bits [9:5]

Indicates the data value register number supplied in the instruction that has been trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IT, bits [4:0]
When ExType == 0b0000 :

Type of the instruction that caused the Guarded Control Stack Data Check Exception.

IT	Meaning
`0b00000`	Guarded Control Stack Data Check Exception is from a procedure return instruction without Pointer authentication.
`0b00001`	Guarded Control Stack Data Check Exception is from a GCSPOPM instruction.
`0b00010`	Guarded Control Stack Data Check Exception is from a procedure return instruction with Pointer authentication that uses key A.
`0b00011`	Guarded Control Stack Data Check Exception is from a procedure return instruction with Pointer authentication that uses key B.
`0b00100`	Guarded Control Stack Data Check Exception is from a GCSSS1 instruction.
`0b00101`	Guarded Control Stack Data Check Exception is from a GCSSS2 instruction.
`0b01000`	Guarded Control Stack Data Check Exception is from a GCSPOPCX instruction.
`0b01001`	Guarded Control Stack Data Check Exception is from a GCSPOPX instruction.

All other values are reserved

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

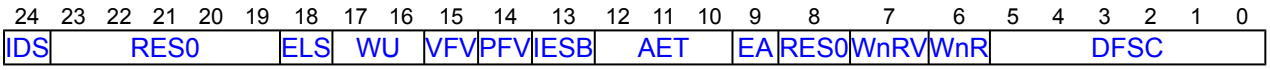
Reserved, RES0.

Additional information for the ISS encoding for a GCS exception

The following fields describe the configuration settings for the traps that are reported using EC value `0b101101` and ExType value `0b0010`:

- GCSCRE0_EL1.STREn
- GCSCR_EL1.STREn.
- GCSCR_EL2.STREn.
- GCSCR_EL3.STREn.
- HFGITR_EL2.nGCSSTR_EL1.

ISS encoding for an SError exception



Note

In earlier versions of the architecture, an SError exception is referred to as an SError interrupt or an asynchronous External abort exception.

IDS, bit [24]

IMPLEMENTATION DEFINED syndrome.

IDS	Meaning
`0b0`	Bits [23:0] of the ISS field holds the fields described in this encoding. **Note** If FEAT_RAS is not implemented, bits [23:0] of the ISS field are RES0.
`0b1`	Bits [23:0] of the ISS field holds IMPLEMENTATION DEFINED syndrome information that can be used to provide additional information about the SError exception.

Note

This field was previously called ISV.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:19]

Reserved, RES0.

ELS, bit [18]

When FEAT_RASv2 is implemented and DFSC == 0b010001:

Meaning of ELR_ELx.

ELS	Meaning
`0b0`	Asynchronous. Does not indicate the trigger for the exception.
`0b1`	Synchronous. The exception was triggered by the instruction at ELR_ELx.

SError exceptions that report this field is 1 are not required to be precise.

The ESR_EL1.AET field describes whether the exception is precise or imprecise.

Corrected, Recoverable or Restartable exceptions are precise. Unrecoverable or Uncontainable exceptions are imprecise.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WU, bits [17:16]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

Write Update. Describes whether a store instruction that generated an External abort updated the location.

WU	Meaning
0b00	Not a store instruction or translation table update, or the location might have been updated.
0b10	Store instruction or translation table update that did not update the location.
0b11	Store instruction or translation table update that updated the location.

In the description of this field, a store instruction is any memory-writing instruction that explicitly performs a store. This includes instructions that both read and write memory.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VFV, bit [15]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

FAR Valid. Indicates the FAR_EL1 register contains a valid virtual address.

VFV	Meaning
0b0	FAR_EL1 is not valid, and holds an UNKNOWN value.
0b1	FAR_EL1 contains a valid virtual address associated with the error.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PFV, bit [14]
When FEAT_PFAR is implemented and DFSC == 0b010001:

PFAR Valid. Describes whether the PFAR_EL1 register is valid.

PFV	Meaning
0b0	PFAR_EL1 is UNKNOWN.
0b1	PFAR_EL1 is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IESB, bit [13]
When FEAT_IESB is implemented and DFSC == 0b010001:

Implicit error synchronization event.

IESB	Meaning
0b0	The SError exception was either not synchronized by the implicit error synchronization event or not taken immediately.
0b1	The SError exception was synchronized by the implicit error synchronization event and taken immediately.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AET, bits [12:10]
When FEAT_RAS is implemented and DFSC == 0b010001:

Asynchronous Error Type.

Describes the PE error state after taking the SError exception.

AET	Meaning
0b000	Uncontainable (UC).
0b001	Unrecoverable state (UEU).
0b010	Restartable state (UEO).
0b011	Recoverable state (UER).
0b110	Corrected (CE).

All other values are reserved.

If multiple errors are taken as a single SError exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA, bit [9]
When FEAT_RAS is implemented and DFSC == 0b010001:

External abort type. Provides an IMPLEMENTATION DEFINED classification of External aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

WnRV, bit [7]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

ESR_EL1.WnR valid.

WnRV	Meaning
0b0	ESR_EL1.WnR is not valid and has been set to 0b0.
0b1	ESR_EL1.WnR is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WnR, bit [6]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

Write-not-Read. When the WnRV field is 0b1, indicates whether an exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

WnR	Meaning
0b0	Exception was caused by an instruction reading from a memory location.
0b1	Exception was caused by an instruction writing to a memory location.

Accessing this bit has the following behavior:

- This bit is RES0 if ESR_EL1.WnRV==0b0.
- This bit is not valid and reads UNKNOWN if an External abort on a Atomic access, reported with ESR_EL1.WU == 0b00.
- Otherwise RW.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DFSC, bits [5:0]

When FEAT_RAS is implemented:

Data Fault Status Code.

DFSC	Meaning
0b000000	Uncategorized error.
0b010001	Asynchronous SError exception.

All other values are reserved.

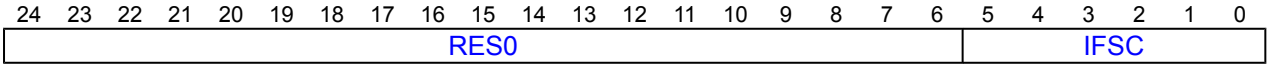
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ISS encoding for an exception from a Breakpoint or Vector Catch debug exception



Bits [24:6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

IFSC	Meaning
0b100010	Debug exception.

The reset behavior of this field is:

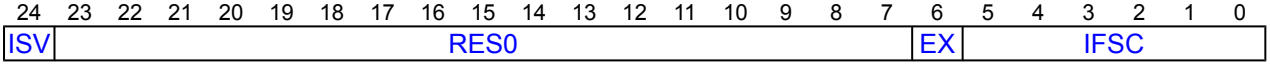
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a Breakpoint or Vector Catch debug exception

For more information about generating these exceptions:

- For exceptions from AArch64, see 'Breakpoint exceptions'.
- For exceptions from AArch32, see 'Breakpoint exceptions' and 'Vector Catch exceptions'.

ISS encoding for an exception from a Software Step exception



ISV, bit [24]

Instruction syndrome valid. Indicates whether the EX bit, ISS[6], is valid, as follows:

ISV	Meaning
0b0	EX bit is RES0.
0b1	EX bit is valid.

See the EX bit description for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:7]

Reserved, RES0.

EX, bit [6]

Exclusive operation. If the ISV bit is set to 1, this bit indicates whether a Load-Exclusive instruction was stepped.

EX	Meaning
0b0	An instruction other than a Load-Exclusive instruction was stepped.
0b1	A Load-Exclusive instruction was stepped.

If the ISV bit is set to 0, this bit is RES0, indicating no syndrome data is available.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IFSC, bits [5:0]

Instruction Fault Status Code.

IFSC	Meaning
0b100010	Debug exception.

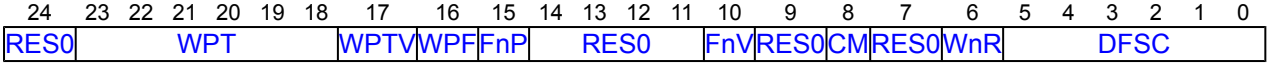
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a Software Step exception

For more information about generating these exceptions, see 'Software Step exceptions'.

ISS encoding for an exception from a Watchpoint exception



Bit [24]

Reserved, RES0.

WPT, bits [23:18]

When FEAT_Debugv8p2 is implemented:

Watchpoint number.

All other values are reserved.

Otherwise:

Reserved, RES0.

WPTV, bit [17]

When FEAT_Debugv8p2 is implemented:

Watchpoint number Valid.

WPTV	Meaning
0b0	The WPT field is invalid, and holds an UNKNOWN value.
0b1	The WPT field is valid, and holds the number of a watchpoint that triggered a Watchpoint exception.

If FEAT_Debugv8p9 is implemented, value 0b0 is not permitted.

When a Watchpoint exception is triggered by a watchpoint match:

- If FEAT_Debugv8p9 is implemented or the PE sets any of FnV, FnP, or WPF to 1, then the PE sets WPTV to 1.
- Otherwise, the PE sets WPTV to an IMPLEMENTATION DEFINED value, 0 or 1.

Otherwise:

Reserved, RES0.

WPF, bit [16]

Watchpoint might be false-positive.

WPF	Meaning	Applies when
0b0	The watchpoint matched an address or address range that was accessed by the instruction.	
0b1	The watchpoint matched an address or address range that might not have been accessed by the instruction.	When FEAT_SVE is implemented or FEAT_SME is implemented

Arm strongly recommends that this bit is set to 0, other than when one of the following instructions might generate a watchpoint match for an address or address range that the instruction does not access:

- An SVE contiguous vector load/store instruction, when the PE is in Streaming SVE mode.
- An SME load/store instruction.

FnP, bit [15]

FAR not Precise.

This field only has meaning if the FAR is valid; that is, when the FnV field is 0. If the FnV field is 1, the FnP field is 0.

FnP	Meaning	Applies when
0b0	If the FnV field is 0, the FAR holds the virtual address of an access or set of contiguous accesses that triggered a Watchpoint exception.	
0b1	The FAR holds any address within the smallest implemented translation granule that contains the virtual address of an access or set of contiguous accesses that triggered a Watchpoint exception.	When FEAT_SVE is implemented or FEAT_SME is implemented

Bits [14:11]

Reserved, RES0.

FnV, bit [10]

FAR not Valid.

FnV	Meaning	Applies when
0b0	The FAR is valid, and its value is as described by the FnP field.	
0b1	The FAR is invalid, and holds an UNKNOWN value.	When FEAT_SVE is implemented or FEAT_SME is implemented

Bit [9]

Reserved, RES0.

CM, bit [8]

Cache maintenance. Indicates whether the Watchpoint exception came from a cache maintenance instruction:

CM	Meaning
0b0	The Watchpoint exception was not generated by the execution of one of the System instructions identified in the description of value 1.
0b1	The Watchpoint exception was generated by the execution of a cache maintenance instruction. The DC ZVA, DC GVA, and DC GZVA instructions are not classified as a cache maintenance instructions, and therefore their execution does not cause this field to be set to 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

WnR, bit [6]

Write not Read. Indicates whether the Watchpoint exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

WnR	Meaning
0b0	Watchpoint exception caused by an instruction reading from a memory location.
0b1	Watchpoint exception caused by an instruction writing to a memory location.

For Watchpoint exceptions on cache maintenance instructions, this bit always returns a value of 1.

For Watchpoint exceptions from an atomic instruction, this field is set to 0 if a read of the location would have generated the Watchpoint exception, otherwise it is set to 1.

If multiple watchpoints match on the same access, it is UNPREDICTABLE which watchpoint generates the Watchpoint exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

DFSC	Meaning
0b100010	Debug exception.

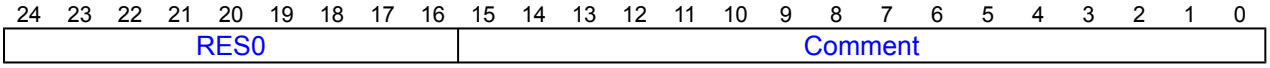
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a Watchpoint exception

For more information about generating these exceptions, see 'Watchpoint exceptions'.

ISS encoding for an exception from execution of a Breakpoint instruction



Bits [24:16]

Reserved, RES0.

Comment, bits [15:0]

Set to the instruction comment field value, zero extended as necessary.

For the AArch32 BKPT instructions, the comment field is described as the immediate field.

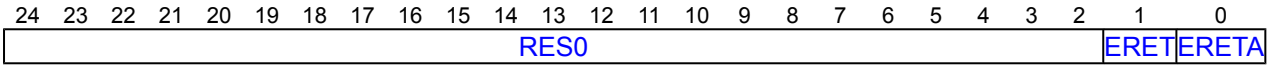
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from execution of a Breakpoint instruction

For more information about generating these exceptions, see 'Breakpoint instruction exceptions'.

ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction



This EC value applies when:

- FEAT_FGT is implemented.
- The Effective value of HCR_EL2.NV is 1.

Bits [24:2]

Reserved, RES0.

ERET, bit [1]

Indicates whether an ERET or ERETA* instruction was trapped to EL1.

ERET	Meaning
0b0	ERET instruction trapped to EL1.
0b1	ERETAA or ERETAB instruction trapped to EL1.

If this bit is 0, the ERETA field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ERETA, bit [0]

Indicates whether an ERETAA or ERETAB instruction was trapped to EL1.

ERETA	Meaning
0b0	ERETAA instruction trapped to EL1.
0b1	ERETAB instruction trapped to EL1.

When the ERET field is 0, this bit is RES0.

The reset behavior of this field is:

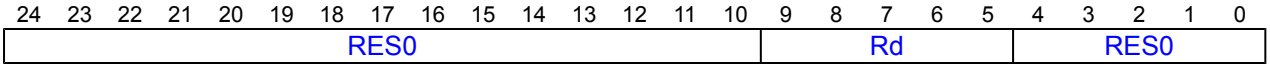
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction

For more information about generating these exceptions, see HCR_EL2.NV.

If FEAT_FGT is implemented, HFGITR_EL2.ERET controls fine-grained trap exceptions from ERET, ERETAA, and ERETAB execution.

ISS encoding for an exception from a TSTART instruction



Bits [24:10]

Reserved, RES0.

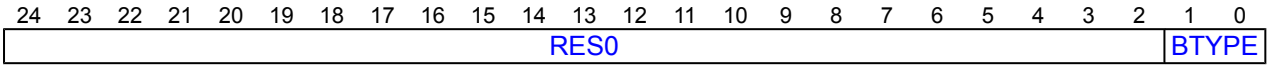
Rd, bits [9:5]

The Rd value from the issued instruction, the general purpose register used for the destination.

Bits [4:0]

Reserved, RES0.

ISS encoding for an exception from Branch Target Identification instruction



Bits [24:2]

Reserved, RES0.

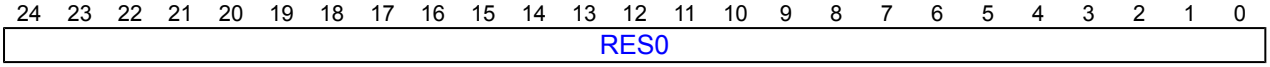
BTYPE, bits [1:0]

This field is set to the PSTATE.BTYPE value that generated the Branch Target Exception.

Additional information for the ISS encoding for an exception from Branch Target Identification instruction

For more information about generating these exceptions, see 'The AArch64 application level programmers model'.

ISS encoding for an exception from a trapped Pointer Authentication instruction



Bits [24:0]

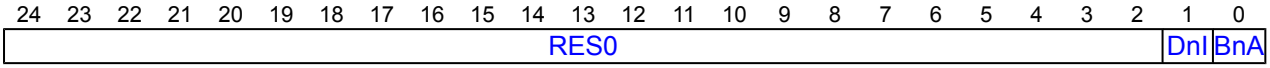
Reserved, RES0.

Additional information for the ISS encoding for an exception from a trapped Pointer Authentication instruction

For more information about generating these exceptions, see:

- HCR_EL2.API, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL2.
- SCR_EL3.API, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL3.

ISS encoding for a PAC Fail exception



Bits [24:2]

Reserved, RES0.

DnI, bit [1]

This field indicates whether the exception is as a result of an Instruction key or a Data key.

DnI	Meaning
0b0	Instruction Key.
0b1	Data Key.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BnA, bit [0]

This field indicates whether the exception is as a result of an A key or a B key.

BnA	Meaning
0b0	A key.
0b1	B key.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for a PAC Fail exception

The following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:

- AUTDA, AUTDZA.

- AUTDB, AUTDZB.
- AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIZA.
- AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZB.
- AUTIASPPC, AUTIASPPCR, AUTIA171615.
- AUTIBSPPC, AUTIBSPPCR, AUTIB171615.

If FEAT_FPACCOMBINE is implemented, the following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:

- RETAA, RETAB.
- RETAASPPC, RETABSPPC.
- RETAASPPCR, RETABSPPCR.
- BLRAA, BLRAAZ, BLRAB, BLRABZ.
- BRAA, BRAB, BRAAZ, BRABZ.
- ERETAA, ERETAB.
- LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

Accessing ESR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name ESR_EL1 or ESR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ESR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ESR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x138];
    else
        X[t, 64] = ESR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = ESR_EL2;
    else
        X[t, 64] = ESR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ESR_EL1;
```

```
MSR ESR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ESR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x138] = X[t, 64];
    else
        ESR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        ESR_EL2 = X[t, 64];
    else
        ESR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ESR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, ESR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x138];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = ESR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = ESR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR ESR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x138] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        ESR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        ESR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

MRS <Xt>, ESR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = ESR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ESR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ESR_EL2;
```

MSR ESR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        ESR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ESR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ESR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ESR_EL2, Exception Syndrome Register (EL2)

The ESR_EL2 characteristics are:

Purpose

Holds syndrome information for an exception taken to EL2.

Configuration

AArch64 System register ESR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HSR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ESR_EL2 are UNDEFINED.

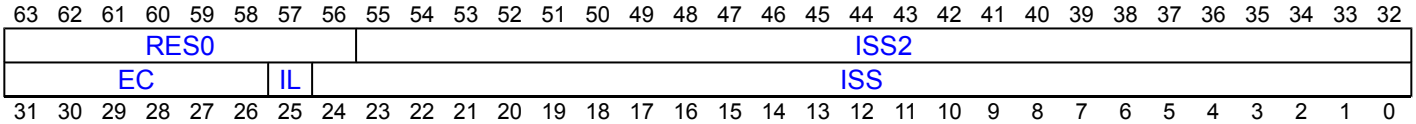
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ESR_EL2 is a 64-bit register.

Field descriptions



ESR_EL2 is made UNKNOWN as a result of an exception return from EL2.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL2, the value of ESR_EL2 is UNKNOWN. The value written to ESR_EL2 must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

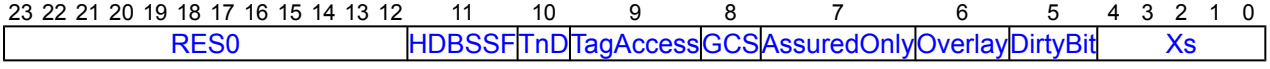
Bits [63:56]

Reserved, RES0.

ISS2, bits [55:32]

ISS2 encoding for an exception, the bit assignments are:

ISS2 encoding for an exception from a Data Abort



Bits [23:12]

Reserved, RES0.

HDBSSF, bit [11]

When FEAT_HDBSS is implemented, IsSecondStage(Fault), and DFSC == 0b0011xx || DFSC == 0b01001x || DFSC == 0b0101xx || DFSC == 0b10001x || DFSC == 0b1001xx:

When DFSC indicates a stage 2 Permission fault, this field indicates whether that fault was caused by the HDBSS being full.

When DFSC indicates a synchronous External abort on translation table walk or hardware update of translation table, this field indicates whether the fault was caused by a write to the HDBSS.

When DFSC indicates a Granule Protection Fault on translation table walk or hardware update of translation table, this field indicates whether the fault was caused by a write to the HDBSS.

HDBSSF	Meaning
0b0	Fault was not caused by HDBSS.
0b1	Fault was caused by HDBSS.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TnD, bit [10]

When FEAT_MTE_CANONICAL_TAGS is implemented:

Tag not Data.

If a memory access generates a Data Abort for a stage 1 Permission fault, this field indicates whether the fault is due to an Allocation Tag access.

TnD	Meaning
0b0	Permission fault is not due to a write of an Allocation Tag to Canonically Tagged memory.
0b1	Permission fault is due to a write of an Allocation Tag to Canonically Tagged memory.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TagAccess, bit [9]

When FEAT_MTE_PERM is implemented:

NoTagAccess fault.

If a memory access generates a Data Abort for a Permission fault, this field indicates whether the fault is due to the NoTagAccess memory attribute.

TagAccess	Meaning
0b0	Permission fault is not due to the NoTagAccess memory attribute.
0b1	Permission fault is due to the NoTagAccess memory attribute.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

GCS, bit [8]
When FEAT_GCS is implemented:

Guarded Control Stack data access.

If a memory access generates a Data Abort, this field indicates whether the fault is due to a Guarded Control Stack data access.

GCS	Meaning
0b0	The Data Abort is not due to a Guarded control stack data access.
0b1	The Data Abort is due to a Guarded control stack data access. The ISV field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]
When FEAT_THE is implemented:

AssuredOnly flag.

If a memory access generates a Data Abort for a stage 2 Permission fault, this field holds information about the fault.

If this field is 1 and ESR_EL2.GCS is 1, then the AssuredOnly check might have been the result of VTCR_EL2.GCSH configuration.

AssuredOnly	Meaning
0b0	The Data Abort is not due to AssuredOnly.
0b1	The Data Abort is due to AssuredOnly.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Overlay, bit [6]
When FEAT_S1POE is implemented or FEAT_S2POE is implemented:

Overlay flag.

If a memory access generates a Data Abort for a Permission fault, this field holds information about the fault.

Overlay	Meaning
0b0	The Data Abort is not due to Overlay Permissions.
0b1	The Data Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]
When FEAT_S1PIE is implemented or FEAT_S2PIE is implemented:

DirtyBit flag.

If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

For any other fault or Access, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Xs, bits [4:0]
When FEAT_LS64 is implemented:

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort exception for a Translation fault, Access flag fault, or Permission fault, then this field holds register specifier, Xs.

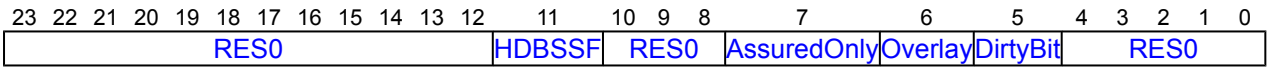
When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort exception for a Translation fault, Access flag fault, or Permission fault, then this field holds register specifier, Xs.

Otherwise, this field is RES0.

Otherwise:

Reserved, RES0.

ISS2 encoding for an exception from an Instruction Abort



Bits [23:12]

Reserved, RES0.

HDBSSF, bit [11]

When FEAT_HDBSS is implemented:

Indicates that the fault was caused by the HDBSS.

When IFSC indicates a Permission fault, this field indicates whether the fault was caused by the HDBSS being full.

When IFSC indicates a synchronous External abort on translation table walk or hardware update of translation table, this field indicates whether the fault was caused by a write to the HDBSS.

When IFSC indicates a Granule Protection Fault on translation table walk or hardware update of translation table, this field indicates whether the fault was caused by a write to the HDBSS.

HDBSSF	Meaning
0b0	Fault was not caused by HDBSS.
0b1	Fault was caused by HDBSS.

This field only applies for stage 2 Permission faults.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [10:8]

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented:

AssuredOnly flag.

If a memory access generates a Instruction Abort for a Permission fault, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Instruction Abort is not due to AssuredOnly.
0b1	Instruction Abort is due to stage 2 AssuredOnly attribute.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When FEAT_S1POE is implemented or FEAT_S2POE is implemented:

Overlay flag.

If a memory access generates a Instruction Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Instruction Abort is not due to Overlay Permissions.
0b1	Instruction Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

When FEAT_S2PIE is implemented:

DirtyBit flag.

If a write access to memory generates an Instruction Abort for a Permission fault using Indirect Permission, this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

For any other fault or Access, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

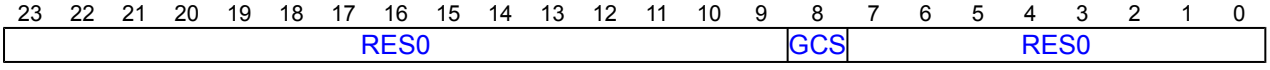
Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

ISS2 encoding for an exception from a Watchpoint exception



Bits [23:9]

Reserved, RES0.

GCS, bit [8]

When FEAT_GCS is implemented:

Guarded control stack data access.

Indicates that the Watchpoint exception is due to a Guarded control stack data access.

GCS	Meaning
0b0	The Watchpoint exception is not due to a Guarded control stack data access.
0b1	The Watchpoint exception is due to a Guarded control stack data access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

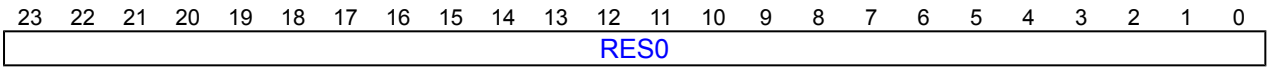
Otherwise:

Reserved, RES0.

Bits [7:0]

Reserved, RES0.

ISS2 encoding for all other exceptions



Bits [23:0]

Reserved, RES0.

EC, bits [31:26]

Exception Class. Indicates the reason for the exception that this register holds information about.

For each EC value, the table references a subsection that gives information about:

- The cause of the exception, for example the configuration required to enable the trap.
- The encoding of the associated ISS.

Possible values of the EC field are:

EC	Meaning	ISS	ISS2	Applies when
0b000000	Unknown reason.	ISS encoding for exceptions with an unknown reason	ISS2 encoding for all other exceptions	
0b000001	Trapped WF* instruction execution. Conditional WF* instructions that fail their condition code check do not cause an exception.	ISS encoding for an exception from a WF* instruction	ISS2 encoding for all other exceptions	
0b000011	Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC value 0b000000.	ISS encoding for an exception from an MCR or MRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000100	Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC value 0b000000.	ISS encoding for an exception from an MCRR or MRRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000101	Trapped MCR or MRC access with (coproc==0b1110).	ISS encoding for an exception from an MCR or MRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000110	Trapped LDC or STC access. The only architected uses of these instruction are: • An STC to write data to memory from DBGDTRRXint. • An LDC to read data from memory to DBGDTRTXint.	ISS encoding for an exception from an LDC or STC instruction	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000111	Access to SME, SVE, Advanced SIMD or floating-point functionality trapped by CPACR_EL1.FPEN, CPTR_EL2.FPEN, CPTR_EL2.TFP, or CPTR_EL3.TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2.TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000.	ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps	ISS2 encoding for all other exceptions	
0b001000	Trapped VMRS access, from ID group trap, that is not reported using EC value 0b000111.	ISS encoding for an exception from an MCR or MRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b001001	Trapped use of a Pointer authentication instruction.	ISS encoding for an exception from a trapped Pointer Authentication instruction	ISS2 encoding for all other exceptions	When FEAT_PAuth is implemented
0b001010	Trapped execution of any instruction not covered by other EC values.	ISS encoding for an exception from any other instruction	ISS2 encoding for all other exceptions	When FEAT_LS64 is implemented, or FEAT_SPEv1p5 is implemented, or

				FEAT_TRBEv1p1 is implemented
0b001100	Trapped MRRC access with (coproc==0b1110).	ISS encoding for an exception from an MCRR or MRRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b001101	Branch Target Exception.	ISS encoding for an exception from Branch Target Identification instruction	ISS2 encoding for all other exceptions	When FEAT_BTI is implemented
0b001110	Illegal Execution state.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault	ISS2 encoding for all other exceptions	
0b010001	SVC instruction execution in AArch32 state. This is reported in ESR_EL2 only when the exception is generated because the value of HCR_EL2.TGE is 1.	ISS encoding for an exception from HVC or SVC instruction execution	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b010010	HVC instruction execution in AArch32 state, when HVC is not disabled.	ISS encoding for an exception from HVC or SVC instruction execution	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b010011	SMC instruction execution in AArch32 state, when SMC is not disabled. This is reported in ESR_EL2 only when the exception is generated because the value of HCR_EL2.TSC is 1.	ISS encoding for an exception from SMC instruction execution in AArch32 state	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b010100	Trapped MSRR, MRRS or System instruction execution in AArch64 state, that is not reported using EC value 0b000000.	ISS encoding for an exception from MSRR, MRRS, or 128-bit System instruction execution in AArch64 state	ISS2 encoding for all other exceptions	When FEAT_SYSREG128 is implemented or FEAT_SYSINSTR128 is implemented
0b010101	SVC instruction execution in AArch64 state.	ISS encoding for an exception from HVC or SVC instruction execution	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b010110	HVC instruction execution in AArch64 state, when HVC is not disabled.	ISS encoding for an exception from HVC or SVC instruction execution	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b010111	SMC instruction execution in AArch64 state, when SMC is not disabled. This is reported in ESR_EL2 only when the exception is generated because the value of HCR_EL2.TSC is 1.	ISS encoding for an exception from SMC instruction execution in AArch64 state	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b011000	Trapped MSR, MRS or System instruction execution in AArch64 state, that is not reported using EC values 0b000000, 0b000001 or 0b000111.	ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented

	This includes all instructions that cause exceptions that are part of the encoding space defined in 'System instruction class encoding overview', except for those exceptions reported using EC values `0b000000`, `0b000001`, or `0b000111`.			
`0b011001`	Access to SVE functionality trapped as a result of CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ, that is not reported using EC value `0b000000`.	ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ	ISS2 encoding for all other exceptions	When FEAT_SVE is implemented
`0b011010`	Trapped ERET, ERETAA, or ERETAB instruction execution.	ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction	ISS2 encoding for all other exceptions	When FEAT_FGT is implemented or FEAT_NV is implemented
`0b011011`	Exception from an access to a TSTART instruction at EL0 when SCTLR_EL1.TME0 == 0, EL0 when SCTLR_EL2.TME0 == 0, at EL1 when SCTLR_EL1.TME == 0, at EL2 when SCTLR_EL2.TME == 0 or at EL3 when SCTLR_EL3.TME == 0.	ISS encoding for an exception from a TSTART instruction	ISS2 encoding for all other exceptions	When FEAT_TME is implemented
`0b011100`	Exception from a PAC Fail	ISS encoding for a PAC Fail exception	ISS2 encoding for all other exceptions	When FEAT_FPAC is implemented
`0b011101`	Access to SME functionality trapped as a result of CPACR_EL1.SMEN, CPTR_EL2.SMEN, CPTR_EL2.TSM, CPTR_EL3.ESM, or an attempted execution of an instruction that is illegal because of the value of PSTATE.SM or PSTATE.ZA, that is not reported using EC value `0b000000`.	ISS encoding for an exception due to SME functionality	ISS2 encoding for all other exceptions	When FEAT_SME is implemented
`0b100000`	Instruction Abort from a lower Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.	ISS encoding for an exception from an Instruction Abort	ISS2 encoding for an exception from an Instruction Abort	
`0b100001`	Instruction Abort taken without a change in Exception level.	ISS encoding for an exception from an Instruction Abort	ISS2 encoding for an exception	

	Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.		from an Instruction Abort	
0b100010	PC alignment fault exception.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault	ISS2 encoding for all other exceptions	
0b100100	Data Abort exception from a lower Exception level, excluding Data Abort exceptions taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support. These Data Abort exceptions might be generated from Exception levels in any Execution state. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.	ISS encoding for an exception from a Data Abort	ISS2 encoding for an exception from a Data Abort	
0b100101	Data Abort exception without a change in Exception level, or Data Abort exceptions taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.	ISS encoding for an exception from a Data Abort	ISS2 encoding for an exception from a Data Abort	
0b100110	SP alignment fault exception.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault	ISS2 encoding for all other exceptions	
0b100111	Memory Operation Exception.	ISS encoding for an exception from the Memory Copy and Memory Set instructions	ISS2 encoding for all other exceptions	When FEAT_MOPS is implemented

0b101000	Trapped floating-point exception taken from AArch32 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.	ISS encoding for an exception from a trapped floating-point exception	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b101100	Trapped floating-point exception taken from AArch64 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.	ISS encoding for an exception from a trapped floating-point exception	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b101101	GCS exception.	ISS encoding for a GCS exception	ISS2 encoding for all other exceptions	When FEAT_GCS is implemented
0b101111	SError exception.	ISS encoding for an SError exception	ISS2 encoding for all other exceptions	
0b110000	Breakpoint exception from a lower Exception level.	ISS encoding for an exception from a Breakpoint or Vector Catch debug exception	ISS2 encoding for all other exceptions	
0b110001	Breakpoint exception taken without a change in Exception level.	ISS encoding for an exception from a Breakpoint or Vector Catch debug exception	ISS2 encoding for all other exceptions	
0b110010	Software Step exception from a lower Exception level.	ISS encoding for an exception from a Software Step exception	ISS2 encoding for all other exceptions	
0b110011	Software Step exception taken without a change in Exception level.	ISS encoding for an exception from a Software Step exception	ISS2 encoding for all other exceptions	
0b110100	Watchpoint from a lower Exception level, excluding Watchpoint Exceptions taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support. These Watchpoint Exceptions might be generated from Exception	ISS encoding for an exception from a Watchpoint exception	ISS2 encoding for an exception from a Watchpoint exception	

	levels using any Execution state.			
0b110101	Watchpoint exceptions without a change in Exception level, or Watchpoint exceptions taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support.	ISS encoding for an exception from a Watchpoint exception	ISS2 encoding for an exception from a Watchpoint exception	
0b111000	BKPT instruction execution in AArch32 state.	ISS encoding for an exception from execution of a Breakpoint instruction	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b111010	Vector Catch exception from AArch32 state. The only case where a Vector Catch exception is taken to an Exception level that is using AArch64 is when the exception is routed to EL2 and EL2 is using AArch64.	ISS encoding for an exception from a Breakpoint or Vector Catch debug exception	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b111100	BRK instruction execution in AArch64 state.	ISS encoding for an exception from execution of a Breakpoint instruction	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b111101	Profiling exception	ISS encoding for a Profiling exception	ISS2 encoding for all other exceptions	When FEAT_EBEP is implemented, or FEAT_SPE_EXC is implemented, or FEAT_TRBE_EXC is implemented

All other EC values are reserved by Arm, and:

- Unused values in the range 0b000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
- Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [25]

Instruction Length for synchronous exceptions. Possible values of this bit are:

IL	Meaning
0b0	16-bit instruction trapped.
0b1	32-bit instruction trapped. This value is also used when the exception is one of the following: • An SError exception. • An Instruction Abort exception. • A PC alignment fault exception. • An SP alignment fault exception. • A Data Abort exception for which the value of the ISV bit is 0. • An Illegal Execution state exception. • Any debug exception except for Breakpoint instruction exceptions. For Breakpoint instruction exceptions, this bit has its standard meaning: ◦ 0b0: 16-bit T32 BKPT instruction. ◦ 0b1: 32-bit A32 BKPT instruction or A64 BRK instruction. • An exception reported using EC value 0b000000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [24:0]

Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

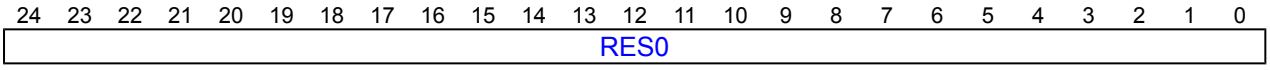
Typically, an ISS encoding has a number of subfields. When an ISS subfield holds a register number, the value returned in that field is the AArch64 view of the register number.

For an exception taken from AArch32 state, see 'Mapping of the general-purpose registers between the Execution states'.

If the AArch32 register descriptor is 0b1111, then:

- If the instruction that generated the exception was not UNPREDICTABLE, the field takes the value 0b11111.
- If the instruction that generated the exception was UNPREDICTABLE, the field takes an UNKNOWN value that must be either:
  - The AArch64 view of the register number of a register that might have been used at the Exception level from which the exception was taken.
  - The value 0b11111.

ISS encoding for exceptions with an unknown reason



Bits [24:0]

Reserved, RES0.

Additional information for the ISS encoding for exceptions with an unknown reason

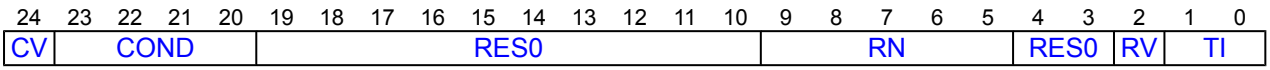
When an exception is reported using this EC value, the IL field is set to 1.

This EC value is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or that is not accessible at the current Exception level and Security state, including:
  - A read access using a System register pattern that is not allocated for reads or that does not permit reads at the current Exception level and Security state.
  - A write access using a System register pattern that is not allocated for writes or that does not permit writes at the current Exception level and Security state.
  - Instruction encodings that are unallocated.
  - Instruction encodings for instructions or System registers that are not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
- In AArch32 state, attempted execution of a short vector floating-point instruction.

- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the SCTLR_EL1.{ITD, SED, CP15BEN} control bits.
- Attempted execution of:
  - An HVC instruction when disabled by HCR_EL2.HCD or SCR_EL3.HCE.
  - An SMC instruction when disabled by SCR_EL3.SMD.
  - An HLT instruction when disabled by EDSCR.HDE.
- Attempted execution of an MSR or MRS instruction to access SP_EL0 when the value of SPSel.SP is 0.
- Attempted execution of an MSR or MRS instruction using a _EL12 register name when the Effective value of HCR_EL2.E2H is not 1.
- Attempted execution, in Debug state, of:
  - A DCPS1 instruction when the value of HCR_EL2.TGE is 1 and EL2 is disabled or not implemented in the current Security state.
  - A DCPS2 instruction from EL1 or EL0 when EL2 is disabled or not implemented in the current Security state.
  - A DCPS3 instruction when the value of EDSCR.SDD is 1, or when EL3 is not implemented.
- When EL3 is using AArch64, attempted execution from Secure EL1 of an SRS instruction using R13_mon.
- In Debug state when the value of EDSCR.SDD is 1, the attempted execution at EL2, EL1, and EL0 of an instruction that is configured to trap to EL3.
- In AArch32 state, the attempted execution of an MRS (banked register) or an MSR (banked register) instruction to SPSR_mon, SP_mon, or LR_mon.
- An exception that is taken to EL2 because the value of HCR_EL2.TGE is 1. If the value of HCR_EL2.TGE is 0, this exception is reported using an ESR_EL2.EC value of `0b000111`.
- In Non-transactional state, attempted execution of a TCOMMIT instruction.

ISS encoding for an exception from a WF* instruction



CV, bit [24]

Condition code valid.

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to `0b1110`.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

        - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
        - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
    - For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:10]

Reserved, RES0.

RN, bits [9:5]

When FEAT_WFxT is implemented:

Register Number. Indicates the register number supplied for a `WFET` or `WFIT` instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

RV, bit [2]

When FEAT_WFxT is implemented:

Register field Valid.

If TI[1] == 1, then this field indicates whether RN holds a valid register number for the register argument to the trapped `WFET` or `WFIT` instruction.

RV	Meaning
`0b0`	Register field invalid.
`0b1`	Register field valid.

If TI[1] == 0, then this field is RES0.

This field is set to 1 on a trap on `WFET` or `WFIT`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TI, bits [1:0]

Trapped instruction. Possible values of this bit are:

TI	Meaning	Applies when
0b00	WFI trapped.	
0b01	WFE trapped.	
0b10	WFIT trapped.	When FEAT_WFxT is implemented
0b11	WFET trapped.	When FEAT_WFxT is implemented

When FEAT_WFxT is implemented, this is a two bit field as shown. Otherwise, bit[1] is RES0.

The reset behavior of this field is:

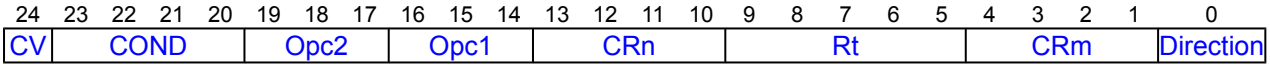
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a WF* instruction

The following fields describe configuration settings for generating this exception:

- HCR.{TWE, TWI}.
- SCTLR_EL1.{nTWE, nTWI}.
- SCTLR_EL2.{nTWE, nTWI}.
- HCR_EL2.{TWE, TWI}.
- SCR_EL3.{TWE, TWI}.

ISS encoding for an exception from an MCR or MRC access



CV, bit [24]

Condition code valid.

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.

        - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
    - For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc2, bits [19:17]

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value `0b000`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [16:14]

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value `0b111`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

If the Rt value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
    - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
    - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value 0b0000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write to System register space. MCR instruction.
0b1	Read from System register space. MRC or VMRS instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an MCR or MRC access

The following fields describe configuration settings for generating exceptions from an MCR or MRC access using coproc 0b1111, that are reported using EC value 0b000011:

- If FEAT_TIDCP1 is implemented, SCTLR_EL1.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch32 state, trapped to EL1.
- CNTKCTL_EL1.{EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- PMUSERENR_EL0.{ER, CR, SW, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- AMUSERENR_EL0.EN, for accesses to Activity Monitors registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- HCR.{TRVM, TVM} and HCR_EL2.{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, trapped to EL2.
- HCR.TTLB and HCR_EL2.TTLB, for execution of TLB maintenance instructions at EL1 using AArch32 state, trapped to EL2.
- HCR.{TSW, TPC, TPU} and HCR_EL2.{TSW, TPC, TPU} for execution of cache maintenance instructions at EL0 and EL1 using AArch32 state, trapped to EL2.
- HCR.TAC and HCR_EL2.TACR, for accesses to the Auxiliary Control Register at EL1 using AArch32 state, trapped to EL2.
- HCR.TIDCP and HCR_EL2.TIDCP, for accesses to lockdown, DMA, and TCM operations at EL0 and EL1 using AArch32 state, trapped to EL2.
- If FEAT_TIDCP1 is implemented, SCTLR_EL2.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch32 state, trapped to EL2.
- HCR.{TID1, TID2, TID3} and HCR_EL2.{TID1, TID2, TID3}, for accesses to ID registers at EL0 and EL1 using AArch32 state, trapped to EL2.
- HCR2.TERR, for Non-secure accesses to error record registers at EL1 using AArch32 state, trapped to EL2.
- HCPTR.TCPAC and CPTR_EL2.TCPAC, for accesses to CPACR_EL1 or CPACR using AArch32 state, trapped to EL2.
- HSTR.T<n> and HSTR_EL2.T<n>, for accesses to System registers using AArch32 state, trapped to EL2.
- CNTHCTL.PL1PCEN and CNTHCTL_EL2.EL1PCEN, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HDCR.TTRF, for Non-secure accesses to trace filter control registers from system registers using AArch32 state, trapped to EL2.
- HDCR.{TPM, TPMCR} and MDCR_EL2.{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HCPTR.TAM and CPTR_EL2.TAM, for accesses to Activity Monitors registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- CPTR_EL3.TCPAC, for accesses to CPACR from EL1 and EL2, and accesses to HCPTR from EL2 using AArch32 state, trapped to EL3.
- MDCR_EL3.TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- CPTR_EL3.TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- If FEAT_FGT is implemented, access to some registers at EL0, trapped to EL2.

The following fields describe configuration settings for generating exceptions from an MCR or MRC access using coproc 0b1110, that are reported using EC value 0b000101:

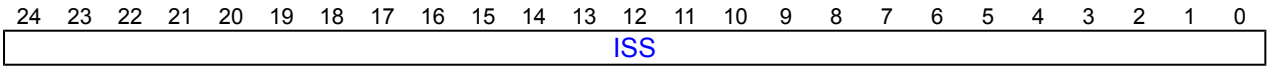
- CPACR_EL1.TTA for accesses to trace registers, trapped to EL1 or EL2.

- MDSCR_EL1.TDCC, for accesses to the Debug Communications Channel (DCC) registers at EL0 and EL1 using AArch32 state, trapped to EL1 or EL2.
- If FEAT_FGT is implemented, MDCR_EL2.TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and MDCR_EL3.TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- HCR.TID0 and HCR_EL2.TID0, for accesses to the JIDR register in the ID group 0 at EL0 and EL1 using AArch32, trapped to EL2.
- HCPTR.TTA and CPTR_EL2.TTA, for accesses to trace registers using AArch32, trapped to EL2.
- HDCR.TDRA and MDCR_EL2.TDRA, for accesses to Debug ROM registers DBGDRAR and DBGDSAR using AArch32, trapped to EL2.
- HDCR.TDOSA and MDCR_EL2.TDOSA, for accesses to powerdown debug registers, using AArch32 state, trapped to EL2.
- HDCR.TDA and MDCR_EL2.TDA, for accesses to other debug registers, using AArch32 state, trapped to EL2.
- CPTR_EL3.TTA, for accesses to trace registers using AArch32, trapped to EL3.
- MDCR_EL3.TDOSA, for accesses to powerdown debug registers using AArch32, trapped to EL3.
- MDCR_EL3.TDA, for accesses to other debug registers, using AArch32, trapped to EL3.

The following fields describe configuration settings for generating exceptions from a VMSR or VMRS access, that are reported using EC value `0b001000`:

- HCR.TID0 and HCR_EL2.TID0, for accesses to the FPSID register in ID group 0 at EL1 using AArch32 state, VMRS access trapped to EL2.
- HCR.TID3 and HCR_EL2.TID3, for accesses to registers in ID group 3 including MVFR0, MVFR1 and MVFR2, VMRS access trapped to EL2.
- HCPTR.{TCP10, TCP11}, for Non-secure accesses to FPSCR, FPSID, FPEXC, MVFR0, MVFR1, and MVFR2, trapped to EL2.

ISS encoding for an exception from any other instruction

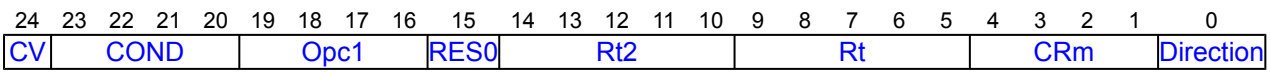


ISS, bits [24:0]

ISS	Meaning	Applies when
`0b00000000000000000000000000`	`ST64BV` instruction trapped.	When FEAT_LS64_V is implemented
`0b00000000000000000000000001`	`ST64BV0` instruction trapped.	When FEAT_LS64_ACCDATA is implemented
`0b00000000000000000000000010`	`LD64B` or `ST64B` instruction trapped.	When FEAT_LS64 is implemented
`0b00000000000000000000000011`	`TSB CSYNC` instruction trapped.	When FEAT_TRBEv1p1 is implemented
`0b00000000000000000000000100`	`PSB CSYNC` instruction trapped.	When FEAT_SPEv1p5 is implemented

All other values are reserved.

ISS encoding for an exception from an MCRR or MRRC access



CV, bit [24]

Condition code valid.

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [19:16]

The Opc1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

Rt2, bits [14:10]

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer.

If the Rt2 value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt2 value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b11111.

- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
    - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
    - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer.

If the Rt value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
    - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
    - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
`0b0`	Write to System register space. MCRR instruction.
`0b1`	Read from System register space. MRRC instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an MCRR or MRRC access

The following fields describe configuration settings for generating exceptions from an MCRR or MRRC access using coproc `0b1111`, that are reported using EC value `0b000100`:

- CNTKCTL_EL1.{EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- PMUSERENR_EL0.{CR, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, trapped to EL1 or EL2.

- AMUSERENR_EL0.{EN}, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 using AArch32 state, trapped to EL1 or EL2.
- HCR.{TRVM, TVM} and HCR_EL2.{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, trapped to EL2.
- HCR2.TERR, for Non-secure accesses to error record registers at EL1 using AArch32 state, trapped to EL2.
- HSTR.T<n> and HSTR_EL2.T<n>, for accesses to System registers using AArch32 state, trapped to EL2.
- CNTHCTL.{PL1PCEN, PL1PCTEN} and CNTHCTL_EL2.{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HDCR.TPM and MDCR_EL2.{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HCPTR.TAM and CPTR_EL2.TAM, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 and EL1 using AArch32 state, trapped to EL2.
- MDCR_EL3.TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- CPTR_EL3.TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- If FEAT_FGT is implemented, HDFGRTR_EL2.PMCCNTR_EL0 for MRRC access and HDFGWTR_EL2.PMCCNTR_EL0 for MCRR access to PMCCNTR at EL0, trapped to EL2.

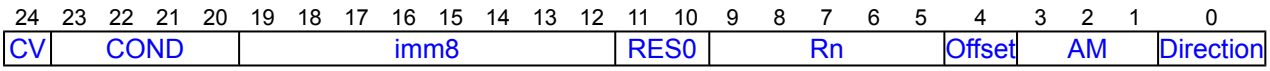
The following fields describe configuration settings for generating exceptions from an MCRR or MRRC access using coproc `0b1110`, that are reported using EC value `0b001100`:

- MDSCR_EL1.TDCC, for accesses to the Debug ROM registers DBGDSAR and DBGDRAR at EL0 using AArch32 state, trapped to EL1 or EL2.
- HDCR.TDRA and MDCR_EL2.TDRA, for accesses to Debug ROM registers DBGDRAR and DBGDSAR using AArch32, trapped to EL2.
- MDCR_EL3.TDA, for accesses to debug registers, using AArch32, trapped to EL3.
- CPACR_EL1.TTA for accesses to trace registers using AArch32, trapped to EL1 or EL2.
- HCPTR.TTA and CPTR_EL2.TTA, for accesses to trace registers using AArch32, trapped to EL2.
- CPTR_EL3.TTA, for accesses to trace registers using AArch32, trapped to EL3.

Note

If the Armv8-A architecture is implemented with an ETMv4 implementation, MCRR and MRRC accesses to trace registers are UNDEFINED and the resulting exception is higher priority than an exception due to these traps.

ISS encoding for an exception from an LDC or STC instruction



CV, bit [24]

Condition code valid.

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to `0b1110`.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

imm8, bits [19:12]

The immediate value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

Rn, bits [9:5]

The Rn value from the issued instruction, the general-purpose register used for the transfer.

If the Rn value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rn value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
  - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
  - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

This field is valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction. When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Offset, bit [4]

Indicates whether the offset is added or subtracted:

Offset	Meaning
0b0	Subtract offset.
0b1	Add offset.

This bit corresponds to the U bit in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AM, bits [3:1]

Addressing mode. The permitted values of this field are:

AM	Meaning
0b000	Immediate unindexed.
0b001	Immediate post-indexed.
0b010	Immediate offset.
0b011	Immediate pre-indexed.
0b100	For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved.
0b110	For a trapped STC instruction, this encoding is reserved.

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in System and memory-mapped registers and translation table entries'.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write to memory. STC instruction.
0b1	Read from memory. LDC instruction.

The reset behavior of this field is:

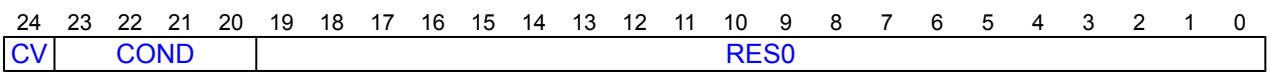
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an LDC or STC instruction

The following fields describe the configuration settings from an LDC or STC access for the traps that are reported using EC value 0b000110:

- MDSCR_EL1.TDCC, for accesses to DBGDTRTXint and DBGDTRRXint, using AArch32 state, trapped to EL1 or EL2.
- HDCR.TDA and MDCR_EL2.TDA, for accesses to DBGDTRTXint and DBGDTRRXint, using AArch32 state, trapped to EL2.
- MDCR_EL3.TDA, for accesses to DBGDTRTXint and DBGDTRRXint, using AArch32 state, trapped to EL3.
- If FEAT_FGT is implemented, MDCR_EL2.TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and MDCR_EL3.TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.

ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps



The accesses covered by this trap include:

- Execution of SVE or Advanced SIMD and floating-point instructions.
- Accesses to the Advanced SIMD and floating-point System registers.
- Execution of SME instructions.

For an implementation that does not include either SVE or support for Advanced SIMD and floating-point, the exception is reported using the EC value `0b000000`.

CV, bit [24]

Condition code valid.

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to `0b1110`.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:0]

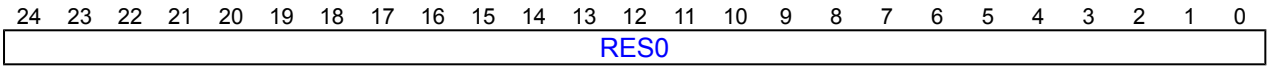
Reserved, RES0.

Additional information for the ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps

The following fields describe the configuration settings for the traps that are reported using EC value `0b000111`:

- HCPTR.{TCP10, TCP11}, for Non-secure accesses to Advanced SIMD and floating-point registers and instructions, trapped to EL2.
- HCPTR.TASE, for Non-secure accesses to Advanced SIMD functionality, trapped to EL2.
- CPACR_EL1.FPEN, for accesses to SIMD and floating-point registers trapped to EL1.
- CPTR_EL2.FPEN and CPTR_EL2.TFP, for accesses to SIMD and floating-point registers trapped to EL2.
- CPTR_EL3.TFP, for accesses to SIMD and floating-point registers trapped to EL3.

ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ



The accesses covered by this trap include:

- Execution of SVE instructions when the PE is not in Streaming SVE mode.
- Accesses to the SVE System registers, ZCR_ELx.

For an implementation that does not include SVE, the exception is reported using the EC value `0b000000`.

Bits [24:0]

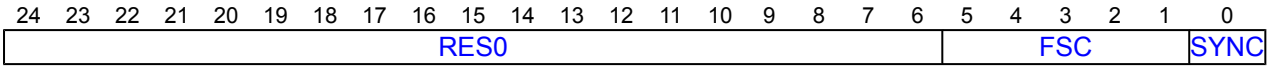
Reserved, RES0.

Additional information for the ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ

The following fields describe the configuration settings for the traps that are reported using EC value `0b011001`:

- CPACR_EL1.ZEN, for execution of SVE instructions and accesses to SVE registers at EL0 or EL1, trapped to EL1.
- CPTR_EL2.ZEN and CPTR_EL2.TZ, for execution of SVE instructions and accesses to SVE registers at EL0, EL1, or EL2, trapped to EL2.
- CPTR_EL3.EZ, for execution of SVE instructions and accesses to SVE registers from all Exception levels, trapped to EL3.

ISS encoding for a Profiling exception



Bits [24:6]

Reserved, RES0.

FSC, bits [5:1]

Indicates why the Profiling exception was generated.

FSC	Meaning	Applies when
0b00000	PMU Profiling exception. One of the following applies, and ESR_EL2.SYNC describes which: • The exception was generated because at least one PMU counter overflow status flag was 1, and was taken asynchronously. • The exception was generated because FEAT_SEBEP is implemented and PSTATE.PPEND was 1, and was taken synchronously.	When FEAT_EBEP is implemented
0b00001	Profiling Buffer management event. The exception was generated because PMBSR_EL2.S was 1.	When FEAT_SPE_EXC is implemented
0b00010	Trace buffer management event. The exception was generated because TRBSR_EL2.IRQ was 1.	When FEAT_TRBE_EXC is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SYNC, bit [0]

Indicates whether the Profiling exception was taken synchronously or asynchronously.

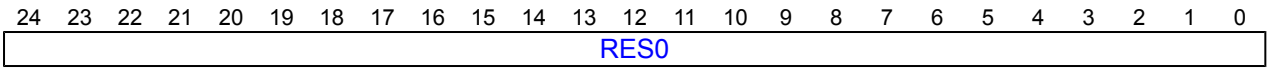
SYNC	Meaning	Applies when
0b0	The exception was taken asynchronously.	
0b1	The exception was taken synchronously.	When FEAT_SEBEP is implemented

If ESR_EL2.FSC does not indicate a PMU Profiling exception, or FEAT_SEBEP is not implemented, then the only permitted value is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault



Bits [24:0]

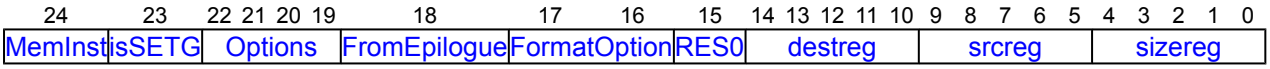
Reserved, RES0.

Additional information for the ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault

There are no configuration settings for generating Illegal Execution state exceptions and PC alignment fault exceptions. For more information about PC alignment fault exceptions, see 'PC alignment checking'.

'SP alignment checking' describes the configuration settings for generating SP alignment fault exceptions.

ISS encoding for an exception from the Memory Copy and Memory Set instructions



MemInst, bit [24]

Indicates the memory instruction class causing the exception.

MemInst	Meaning
0b0	CPYFE*, CPYFM*, CPYE*, and CPYM* instructions.
0b1	SETE*, SETM*, SETGE*, and SETGM* instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

isSETG, bit [23]

Indicates whether the instruction belongs to SETGM* or SETGE* class of instruction.

isSETG	Meaning
0b0	Not a SETGM* or SETGE* instruction.
0b1	SETGM* or SETGE* instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Options, bits [22:19]

Options : the Options field of the instruction.

For Memory Copy instructions, bits[22:19] forms the Options field, which holds the bits[15:12] of the instruction.

For Memory Set instructions:

- Bits[22:21] are RES0.
- Bits[20:19] form the Options field, which holds the bits[13:12] of the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FromEpilogue, bit [18]

Indicates whether the instruction belongs to the epilogue class of Memory Copy or Memory Set instructions.

FromEpilogue	Meaning
0b0	Not an epilogue instruction.
0b1	CPYE*, CPYFE*, SETE*, or SETGE* instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FormatOption, bits [17:16]

Reports the Option used to encode the initial Xs, Xd, and Xn register values provided to the instruction that generated the exception.

FormatOption	Meaning
0b00	Option B.
0b01	Option A.
0b10	Option A.
0b11	Option B.

For more information, see Memory Copy and Memory Set exceptions.

Note

This field was previously presented as two separate bits, WrongOption, bit[17] and OptionA, bit [16], which were already expected to be used together and not individually.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

destreg, bits [14:10]

The destination register value from the issued instruction, containing the destination address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

srcreg, bits [9:5]

The source register value from the issued instruction, containing either the source address or the source data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

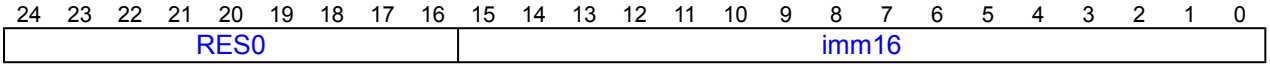
sizereg, bits [4:0]

The size register value from the issued instruction, containing the number of bytes to be transfered or set.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from HVC or SVC instruction execution



Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, and for an A64 SVC instruction, this is the value of the imm16 field of the issued instruction.

For an A32 or T32 SVC instruction:

- If the instruction is unconditional, then:
  - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
  - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from HVC or SVC instruction execution

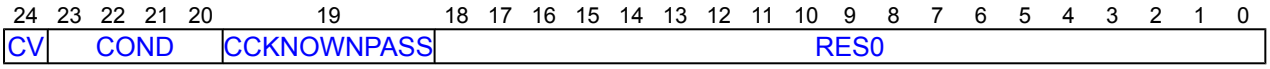
In AArch32 state, the HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.

For T32 and A32 instructions, see 'SVC' and 'HVC'.

For A64 instructions, see 'SVC' and 'HVC'.

If FEAT_FGT is implemented, HFGITR_EL2.{SVC_EL1, SVC_EL0} control fine-grained traps on SVC execution.

ISS encoding for an exception from SMC instruction execution in AArch32 state



For an SMC instruction that completes normally and generates an exception that is taken to EL3, the ISS encoding is RES0.

For an SMC instruction that is trapped to EL2 from EL1 because HCR_EL2.TSC is 1, the ISS encoding is as shown in the diagram.

CV, bit [24]

Condition code valid.

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:

        - With COND set to 0b1110, the value for unconditional.
        - With the COND value held in the instruction.
    - When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
        - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
        - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
    - For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CCKNOWNPASS, bit [19]

Indicates whether the instruction might have failed its condition code check.

CCKNOWNPASS	Meaning
0b0	The instruction was unconditional, or was conditional and passed its condition code check.
0b1	The instruction was conditional, and might have failed its condition code check.

Note

In an implementation in which an SMC instruction that fails it code check is not trapped, this field can always return the value 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [18:0]

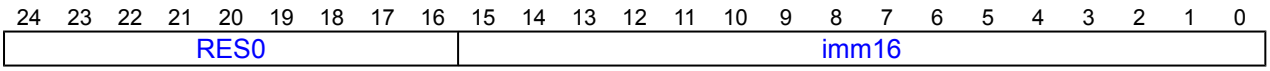
Reserved, RES0.

Additional information for the ISS encoding for an exception from SMC instruction execution in AArch32 state

HCR.TSC describes the configuration settings for trapping SMC instructions to EL2.

HCR_EL2.TSC describes the configuration settings for trapping SMC instructions to EL2.

ISS encoding for an exception from SMC instruction execution in AArch64 state



Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the issued SMC instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

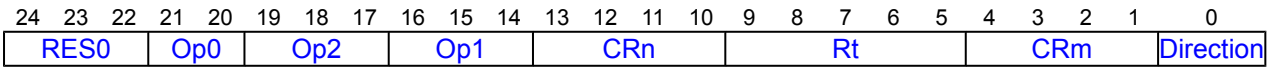
Additional information for the ISS encoding for an exception from SMC instruction execution in AArch64 state

The value of ISS[24:0] described here is used both:

- When an SMC instruction is trapped from EL1 modes.
- When an SMC instruction is not trapped, so completes normally and generates an exception that is taken to EL3.

HCR_EL2.TSC describes the configuration settings for trapping SMC from EL1 modes.

ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state



Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

For system instructions which require that the opcode Rt field is set to `0b11111`, but where the trapped instruction has a different value of Rt, an implementation is permitted to return the value `0b11111`, instead of the value of Rt from the trapped instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write access, including MSR instructions.
0b1	Read access, including MRS instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state

For exceptions caused by System instructions, see 'System instructions' subsection of 'Branches, exception generating and System instructions' for the encoding values returned by an instruction.

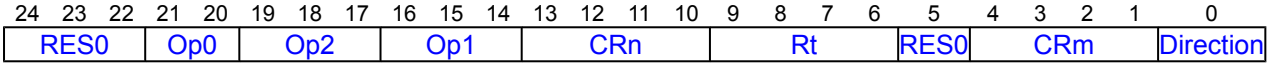
The following fields describe configuration settings for generating the exception that is reported using EC value 0b011000:

- If FEAT_TIDCP1 is implemented, SCTLR_EL1.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch64 state, MSR or MRS access trapped to EL1.
- SCTLR_EL1.UCI, for execution of cache maintenance instructions using AArch64 state, execution is trapped to EL1 or EL2.
- SCTLR_EL1.UCT, for accesses to CTR_EL0 using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- SCTLR_EL1.DZE, for execution of DC ZVA instructions using AArch64 state, execution is trapped to EL1 or EL2.
- SCTLR_EL1.UMA, for accesses to the PSTATE interrupt masks using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- CPACR_EL1.TTA, for accesses to the trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- MDSCR_EL1.TDCC, for accesses to the Debug Communications Channel (DCC) registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- If FEAT_FGT is implemented, MDCR_EL2.TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and MDCR_EL3.TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- CNTKCTL_EL1.{EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN} accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- PMUSERENR_EL0, for accesses to the Performance Monitor registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- AMUSERENR_EL0.EN, for accesses to Activity Monitors registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- HCR_EL2.{TRVM, TVM}, for accesses to virtual memory control registers using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.TDZ, for execution of DC ZVA instructions using AArch64 state, execution is trapped to EL2.
- HCR_EL2.TTLB, for execution of TLB maintenance instructions using AArch64 state, execution is trapped to EL2.
- HCR_EL2.{TSW, TPC, TPU}, for execution of cache maintenance instructions using AArch64 state, execution is trapped to EL2.
- HCR_EL2.TACR, for accesses to the Auxiliary Control Register, ACTLR_EL1, using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.TIDCP, for accesses to lockdown, DMA, and TCM operations using AArch64 state, MSR or MRS access trapped to EL2.
- If FEAT_TIDCP1 is implemented, SCTLR_EL2.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.{TID1, TID2, TID3}, for accesses to ID group 1, ID group 2 or ID group 3 registers, using AArch64 state, MSR or MRS access trapped to EL2.
- If FEAT_MTE2 is implemented, HCR_EL2.TID5, for accesses to GMID_EL1, using AArch64 state, MRS or MSR access at EL1, trapped to EL2.
- If FEAT_IDTE3 is implemented, SCR_EL3.TID3, for accesses to ID group 3 registers using AArch64 state, at EL1 and EL2, trapped to EL3.
- CPTR_EL2.TCPAC, for accesses to CPACR_EL1, using AArch64 state, MSR or MRS access trapped to EL2.
- CPTR_EL2.TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL2.

- MDCR_EL2.TTRF, for accesses to the trace filter control register, TRFCR_EL1, using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.TDRA, for accesses to Debug ROM registers, using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.TDOSA, for accesses to powerdown debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- CNTHCTL_EL2.{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.TDA, for accesses to debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.{TPM, TPMCR}, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL2.
- CPTR_EL2.TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.{NV, NV1}, for Nested virtualization register access, using AArch64 state, MSR or MRS access, trapped to EL2.
- HCR_EL2.AT, for execution of AT S1E* instructions, using AArch64 state, execution is trapped to EL2.
- HCR_EL2.{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access, trapped to EL2.
- SCR_EL3.APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL3.
- SCR_EL3.ST, for accesses to the Counter-timer Physical Secure timer registers, using AArch64 state, MSR or MRS access trapped to EL3.
- SCR_EL3.{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL3.
- CPTR_EL3.TCPAC, for accesses to CPTR_EL2 and CPACR_EL1 using AArch64 state, MSR or MRS access trapped to EL3.
- CPTR_EL3.TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TTRF, for accesses to the trace filter control registers, TRFCR_EL1 and TRFCR_EL2, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TDA, for accesses to debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TDOSA, for accesses to powerdown debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TPM, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL3.
- If FEAT_SPE is implemented:
  - MDCR_EL3.NSPB for accesses to Statistical Profiling and Profiling Buffer control registers, using AArch64 state, MSR or MRS access at EL1 and EL2 trapped to EL3.
  - MDCR_EL2.TPMS for accesses to SPE registers, using AArch64 state, MSR or MRS access at EL1 trapped to EL2.
- CPTR_EL3.TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access, trapped to EL3.
- If FEAT_EVT is implemented, the following registers control traps for EL1 and EL0 Cache controls that use this EC value:
  - HCR_EL2.{TTLBOS, TTLBIS, TICAB, TOCU, TID4}.
  - HCR2.{TTLBIS, TICAB, TOCU, TID4}.
- If FEAT_FGT is implemented:
  - SCR_EL3.FGTEn, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
  - HFGRTR_EL2 for reads and HFGWTR_EL2 for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 trapped to EL2.
  - HFGITR_EL2 for execution of system instructions, MSR or MRS access trapped to EL2.
  - HDFGRTR_EL2 for reads and HDFGWTR_EL2 for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 state trapped to EL2.
  - HAFGRTR_EL2 for reads of Activity Monitor counters, using AArch64 state, MRS access at EL0 and EL1 trapped to EL2.
- If FEAT_RNG_TRAP is implemented, SCR_EL3.TRNDR for reads of RNDR and RNDRRS using AArch64 state, MRS access trapped to EL3.
- If FEAT_SME is implemented:
  - CPTR_EL3.ESM, for MSR or MRS accesses to SMPRI_EL1 at EL1, EL2, and EL3, trapped to EL3.
  - CPTR_EL3.ESM, for MSR or MRS accesses to SMPRIMAP_EL2 at EL2 and EL3, trapped to EL3.
  - SCTLR_EL1.EnTP2, for MSR or MRS accesses to TPIDR2_EL0 at EL0, trapped to EL1 or EL2.
  - SCTLR_EL2.EnTP2, for MSR or MRS accesses to TPIDR2_EL0 at EL0, trapped to EL2.
  - SCR_EL3.EnTP2, for MSR or MRS accesses to TPIDR2_EL0 at EL0, EL1, and EL2, trapped to EL3.
- If FEAT_FPMR is implemented:
  - SCTLR_EL1.EnFPM, for accesses to FPMR at EL0, trapped to EL1 or EL2.
  - SCTLR_EL2.EnFPM, for accesses to FPMR at EL0, trapped to EL2.
  - HCRX_EL2.EnFPM, for accesses to FPMR at EL0 and EL1, trapped to EL2.
  - SCR_EL3.EnFPM, for accesses to FPMR at EL0, EL1, and EL2, trapped to EL3.
- If FEAT_NMI is implemented, HCRX_EL2.TALLINT, for MSR writes of ALLINT at EL1, trapped to EL2.
- If FEAT_FGT2 is implemented:
  - SCR_EL3.FGTEn2, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
  - HFGRTR2_EL2 for reads and HFGWTR2_EL2 for writes of registers, using AArch64 state, using MSR or MRS access at EL1 trapped to EL2.
  - HDFGRTR2_EL2 for reads and HDFGWTR2_EL2 for writes of registers, using AArch64 state, using MSR or MRS access at EL0 and EL1 trapped to EL2.
  - HFGITR2_EL2 for execution of system instructions, MSR or MRS access trapped to EL2.
- If FEAT_ITE is implemented, MDCR_EL3.EnITE, for accesses to Instrumentation trace registers, using AArch64 state, MSR or MRS access, trapped to EL3.
- If FEAT_MEC is implemented, SCR_EL3.MECEn, for accesses to MECID registers at EL2, trapped to EL3.
- If FEAT_SPE_FDS is implemented, MDCR_EL3.EnPMS3 for accesses to SPE registers, using AArch64 state, MSR or MRS access at EL1 and EL2 trapped to EL3.
- If FEAT_SPE_nVM is implemented, MDCR_EL3.EnPMS4 for accesses to SPE registers, using AArch64 state, MSR or MRS access at EL1 and EL2 trapped to EL3.

- If FEAT_RASv2 is implemented, SCR_EL3.TWERR, for accesses to Error Record registers, MSR access at EL1 and EL2 trapped to EL3.
- If FEAT_Debugv8p9 is implemented, MDCR_EL3.EBWE for accesses of MDSELR_EL1, using AArch64 state, MRS or MSR access at EL2 and EL1 trapped to EL3.
- If FEAT_PMUv3p9, FEAT_SPMU, FEAT_EBEP, or FEAT_PMUv3_SS is implemented, MDCR_EL3.EnPM2, for accesses to PMU registers, using AArch64 state, MSR or MRS access at EL2, EL1, and EL0, trapped to EL3.
- If FEAT_PMUv3_SS is implemented, MDCR_EL3.EnPMSS, for accesses to PMU Snapshot registers, using AArch64 state, MSR or MRS access at EL2 and EL1 trapped to EL3.
- If FEAT_THE is implemented, SCR_EL3.RCWMASKEn for accesses to RCWMASK_EL1 and RCWSMASK_EL1, using AArch64 state, MSR or MRS access at EL2 and EL1 trapped to EL3.
- If FEAT_AIE is implemented, SCR_EL3.AIEn for accesses to Extended Memory Attribute registers, MSR or MRS access at EL2 and EL1 trapped to EL3.
- If FEAT_S1PIE, FEAT_S2PIE, FEAT_S1POE, or FEAT_S2POE is implemented, SCR_EL3.PIEn for accesses to Permission Indirection, Overlay registers, MSR or MRS access at EL2, EL1 and EL0 trapped to EL3.
- If FEAT_MPAM_PE_BW_CTRL is implemented, MPAMBW2_EL2.{nTRAP_MPAMBWIDR_EL1, nTRAP_MPAMBW0_EL1, nTRAP_MPAMBW1_EL1} for accesses to MPAMBW*_EL1 registers, using AArch64 state, MRS or MSR access at EL1, trapped to EL2.
- If FEAT_MPAM_PE_BW_CTRL and FEAT_SME are implemented, MPAMBW2_EL2.nTRAP_MPAMBWSM_EL1, for accesses to MPAMBWSM_EL1, using AArch64 state, MRS or MSR access at EL1, trapped to EL2.
- If FEAT_MPAM_PE_BW_CTRL is implemented, MPAMBW3_EL3.nTRAPLOWER, for accesses to MPAMBW_EL2 registers and MPAMBW_EL1 registers, using AArch64 state, MRS or MSR access at EL1, trapped to EL3.
- If FEAT_HACDBS is implemented, SCR_EL3.HACDBSEn, for accesses to HACDBSBR_EL2 and HACDBSCONS_EL2, using AArch64 state, MRS or MSR access at EL2, trapped to EL3.
- If FEAT_HDBSS is implemented, SCR_EL3.HDBSSEn, for accesses to HDBSSBR_EL2 and HDBSSPROD_EL2, using AArch64 state, MRS or MSR access at EL2, trapped to EL3.
- If FEAT_SRMASK is implemented, SCR_EL3.SRMASKEn, for MSR or MRS accesses at EL1 or EL2 to the MASK registers using AArch64 state, trapped to EL3.

ISS encoding for an exception from MSRR, MRRS, or 128-bit System instruction execution in AArch64 state



Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:6]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

Note

This value represents register pair of X[Rt:0], X[Rt:1].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write access, MSRR instructions.
0b1	Read access, MRRS instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

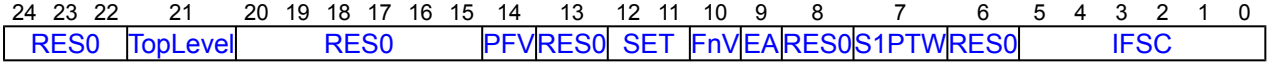
Additional information for the ISS encoding for an exception from MSRR, MRRS, or 128-bit System instruction execution in AArch64 state

The following fields describe configuration settings for generating exceptions from an MSRR or MRRS access that are reported using EC value 0b010100:

- If FEAT_FGT is implemented:
  - HFGRTR_EL2 for reads and HFGWTR_EL2 for writes of registers, using AArch64 state, accesses at EL1 trapped to EL2.
- If FEAT_FGT2 is implemented:
  - HFGRTR2_EL2.nRCWSMASK_EL1 for reads and HFGWTR2_EL2.nRCWSMASK_EL1 for writes of RCWSMASK_EL1, using AArch64 state, accesses at EL1 trapped to EL2.
- If FEAT_SYSREG128 is implemented:
  - SCTLR2_EL1.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL0 trapped to EL1.
  - SCTLR2_EL2.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL0 trapped to EL2.
  - HCRX_EL2.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL1 and EL0 trapped to EL2.

    - SCR_EL3.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL2, EL1, and EL0 trapped to EL3.
- If FEAT_D128 is implemented:
    - HCR_EL2.{TRVM, TVM} for accesses to TTBR0_EL1 and TTBR1_EL1, accesses at EL1 and EL0 trapped to EL2.
    - HCRX_EL2.D128En for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL1 trapped to EL2.
    - SCR_EL3.D128En for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL2 and EL1 trapped to EL3.
- If FEAT_THE is implemented, SCR_EL3.RCWMASKEn for accesses to RCWMASK_EL1 and RCWSMASK_EL1, using AArch64 state, accesses at EL2 and EL1 trapped to EL3.

ISS encoding for an exception from an Instruction Abort



When FEAT_THE is implemented, if a memory access generates an Instruction Abort for AssuredOnly, the ISS2 encoding for an exception from an Instruction Abort includes further information about the exception.

When FEAT_S1POE or FEAT_S2POE is implemented, if a memory access generates an Instruction Abort due to a Permission fault, the ISS2 encoding for an exception from an Instruction Abort includes further information about the exception.

Bits [24:22]

Reserved, RES0.

TopLevel, bit [21]

When FEAT_THE is implemented:

Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [20:15]

Reserved, RES0.

PFV, bit [14]

When FEAT_PFAR is implemented:

PFAR Valid. Describes whether the PFAR_EL2 register is valid.

PFV	Meaning
0b0	PFAR_EL2 is UNKNOWN.
0b1	PFAR_EL2 is valid.

This field is valid only if the IFSC code is 0b10000, 0b01001x, or 0b0101xx.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [13]

Reserved, RES0.

SET, bits [12:11]
When FEAT_RAS is implemented and IFSC == 0b010000:

Synchronous Error Type. When IFSC is `0b010000`, describes the PE error state after taking the Instruction Abort exception.

SET	Meaning	Applies when
`0b00`	Recoverable state (UER).	
`0b10`	Uncontainable (UC).	When FEAT_RASv2 is not implemented
`0b11`	Restartable state (UEO).	

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External abort exception might result in a PE state that is not recoverable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]
When IFSC == 0b010000:

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
`0b0`	FAR is valid.
`0b1`	FAR is not valid, and holds an UNKNOWN value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

IFSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	When FEAT_RAS is not implemented
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented

0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented

All other values are reserved.

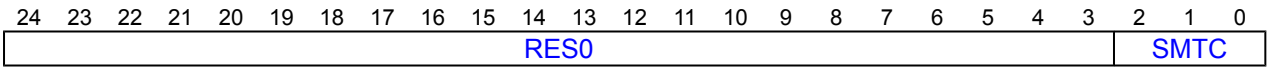
For more information about the lookup level associated with a fault, see 'The lookup level associated with MMU faults'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception due to SME functionality



The accesses covered by this trap include:

- Execution of SME instructions.
- Execution of SVE and Advanced SIMD instructions, when the PE is in Streaming SVE mode.
- Direct accesses of SVCR, SMCR_EL1, SMCR_EL2, SMCR_EL3.

Bits [24:3]

Reserved, RES0.

SMTC, bits [2:0]

SME Trap Code. Identifies the reason for instruction trapping.

SMTC	Meaning	Applies when
0b000	Access to SME functionality trapped as a result of CPACR_EL1.SMEN, CPTR_EL2.SMEN, CPTR_EL2.TSM, or CPTR_EL3.ESM, that is not reported using EC value 0b000000.	
0b001	Advanced SIMD, SVE, or SVE2 instruction trapped because PSTATE.SM is 1.	
0b010	SME instruction trapped because PSTATE.SM is 0.	
0b011	SME instruction trapped because PSTATE.ZA is 0.	
0b100	Access to the SME2 ZT0 register trapped as a result of SMCR_EL1.EZT0, SMCR_EL2.EZT0, or SMCR_EL3.EZT0.	When FEAT_SME2 is implemented

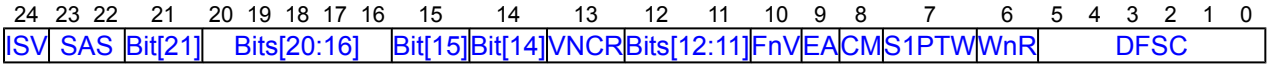
All other values are reserved.

Additional information for the ISS encoding for an exception due to SME functionality

The following fields describe the configuration settings for the traps that are reported using the EC value 0b011101:

- CPACR_EL1.SMEN, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access SVCR and SMCR_EL1 System registers at EL1 and EL0, trapped to EL1 or EL2.
- CPTR_EL2.SMEN and CPTR_EL2.TSM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access SVCR, SMCR_EL1, SMCR_EL2 at EL2, EL1, and EL0, trapped to EL2.
- CPTR_EL3.ESM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access SVCR, SMCR_EL1, SMCR_EL2, SMCR_EL3 from all Exception levels and any Security state, trapped to EL3.
- If FEAT_SME2 is implemented:
  - SMCR_EL1.EZT0, for accesses to ZT0 at EL1 and EL0, trapped to EL1 or EL2.
  - SMCR_EL2.EZT0, for accesses to ZT0 at EL2, EL1, and EL0, trapped to EL2.
  - SMCR_EL3.EZT0, for accesses to ZT0 at any Exception level, trapped to EL3.

ISS encoding for an exception from a Data Abort



The ISS2 encoding for an exception from a Data Abort includes further information about the exception when any of the following features are implemented:

- FEAT_LS64_V.
- FEAT_LS64_ACCDATA.
- FEAT_THE.
- FEAT_S1POE or FEAT_S2POE.
- FEAT_S1PIE or FEAT_S2PIE.
- FEAT_GCS.
- FEAT_MTE_CANONICAL_TAGS.
- FEAT_MTE_PERM.

ISV, bit [24]

Instruction Syndrome Valid. Indicates whether the syndrome information in ISS[23:14] is valid.

ISV	Meaning
0b0	No valid instruction syndrome. ISS[23:14] are RES0.
0b1	ISS[23:14] hold a valid instruction syndrome.

In ESR_EL2, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

In ESR_EL2, ISV is 1 when FEAT_LS64_V is implemented and a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

In ESR_EL2, ISV is 1 when FEAT_LS64_ACCDATA is implemented and a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For other faults reported in ESR_EL2, ISV is 0 except for the following stage 2 aborts:

- AArch64 loads and stores of a single general-purpose register (including the register specified with `0b11111`, including those with Acquire/Release semantics, but excluding Load Exclusive or Store Exclusive and excluding those with writeback).
- AArch32 instructions where the instruction:
    - Is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
    - Is not performing register writeback.
    - Is not using R15 as a source or destination register.

For these stage 2 aborts, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, and otherwise indicates whether ISS[23:14] hold a valid syndrome.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64_V is implemented and a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64_ACCDATA is implemented and a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

When FEAT_RAS is implemented, ISV is 0 for any synchronous External abort.

When FEAT_RAS is not implemented, it is IMPLEMENTATION DEFINED whether ISV is set to 1 or 0 on a synchronous External abort on a stage 2 translation table walk.

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When FEAT_MOPS is implemented, for a synchronous Data Abort on a Memory Copy and Memory Set instruction, ISV is 0.

When FEAT_MTE is implemented, for a synchronous Data Abort on an instruction that directly accesses Allocation Tags, ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SAS, bits [23:22]
When ISV == 1:

Syndrome Access Size. Indicates the size of the access attempted by the faulting operation.

SAS	Meaning
`0b00`	Byte
`0b01`	Halfword
`0b10`	Word
`0b11`	Doubleword

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is `0b11`.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is `0b11`.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is `0b11`.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[21]
When ISV == 1:

SSE, bit [21]

Syndrome Sign Extend. For a byte, halfword, or word load operation, indicates whether the data item must be sign extended.

SSE	Meaning
0b0	Sign-extension not required.
0b1	Data item must be sign-extended.

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

For all other operations, this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ISV == 0 and FEAT_THE is implemented:

TopLevel, bit [21]

Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits[20:16]
When ISV == 1:

SRT, bits [4:0] of bits [20:16]

Syndrome Register Transfer. The register number of the Wt/Xt/Rt operand of the faulting instruction.

If the exception was taken from an Exception level that is using AArch32, then this is the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states'.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ISV == 0, FEAT_RASv2 is implemented, and (DFSC == 0b010000, or DFSC IN {0b01001x}, or DFSC IN {0b0101xx}):

Bits [4:2] of bits [20:16]

Reserved, RES0.

WU, bits [1:0] of bits [20:16]

Write Update. Describes whether a store instruction that generated an External abort updated the location.

WU	Meaning
0b00	Not a store instruction or translation table update, or the location might have been updated.
0b10	Store instruction or translation table update that did not update the location.
0b11	Store instruction or translation table update that updated the location.

In the description of this field, a store instruction is any memory-writing instruction that explicitly performs a store. This includes instructions that both read and write memory.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[15]
When ISV == 1:

SF, bit [15]

Sixty Four bit general-purpose register transfer. Width of the register accessed by the instruction is 64-bit.

SF	Meaning
0b0	Instruction loads/stores a 32-bit general-purpose register.
0b1	Instruction loads/stores a 64-bit general-purpose register.

Note

This field specifies the register width identified by the instruction, not the Execution state.

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ISV == 0:

FnP, bit [15]

FAR not Precise.

FnP	Meaning	Applies when
0b0	The FAR holds the faulting virtual address that generated the Data Abort.	
0b1	The FAR holds any virtual address within the naturally-aligned granule that contains the faulting virtual address that generated a Data Abort due to an SVE contiguous vector load/store instruction, or an SME load/store instruction. For more information about the naturally-aligned fault granule, see FAR_ELx (for example, FAR_EL1).	When FEAT_SME is implemented or FEAT_SVE is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[14]
When ISV == 1:

AR, bit [14]

Acquire/Release.

AR	Meaning
0b0	Instruction did not have acquire/release semantics.
0b1	Instruction did have acquire/release semantics.

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_PFAR is implemented, ISV == 0, and (DFSC == 0b010000, or DFSC IN {0b01001x}, or DFSC IN {0b0101xx}):

PFV, bit [14]

PFAR Valid. Describes whether the PFAR_EL2 register is valid.

PFV	Meaning
0b0	PFAR_EL2 is UNKNOWN.
0b1	PFAR_EL2 is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VNCR, bit [13]

Indicates that the fault came from use of VNCR_EL2 register by EL1 code.

VNCR	Meaning	Applies when
0b0	The fault was not generated by the use of VNCR_EL2 by EL1 code.	
0b1	The fault was generated by the use of VNCR_EL2 by EL1 code.	When FEAT_NV2 is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits[12:11]
When (DFSC IN {0b00xxxx} || DFSC IN {0b10101x}) && !(DFSC IN {0b0000xx}):

LST, bits [1:0] of bits [12:11]

Load/Store Type. Used when a Translation fault, Access flag fault, or Permission fault generates a Data Abort.

LST	Meaning	Applies when
0b00	The instruction that generated the Data Abort is not specified by this field.	
0b01	An ST64BV instruction generated the Data Abort.	When FEAT_LS64_V is implemented
0b10	An LD64B or ST64B instruction generated the Data Abort.	When FEAT_LS64 is implemented
0b11	An ST64BV0 instruction generated the Data Abort.	When FEAT_LS64_ACCDATA is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_RAS is implemented and (DFSC == 0b010000, or DFSC IN {0b01001x}, or DFSC IN {0b0101xx}):

SET, bits [1:0] of bits [12:11]

Synchronous Error Type. Used when a synchronous External abort, not on a Translation table walk or hardware update of the Translation table, generated the Data Abort. Describes the PE error state after taking the Data Abort exception.

SET	Meaning	Applies when
0b00	Recoverable state (UER).	
0b10	Uncontainable (UC). If FEAT_RASv2 is implemented, this value is reserved.	When FEAT_RASv2 is implemented
0b11	Restartable state (UEO).	

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External abort exception might result in a PE state that is not recoverable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	FAR is valid.
0b1	FAR is not valid, and holds an UNKNOWN value.

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

CM	Meaning
0b0	The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1.
0b1	The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA, DC GVA, and DC GZVA instructions are not classified as cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

WnR	Meaning
0b0	Abort caused by an instruction reading from a memory location.
0b1	Abort caused by an instruction writing to a memory location.

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- If FEAT_RASv2 is implemented, an External abort on an Atomic access, reported with ESR_EL2.WU set to 0b00.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

DFSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Synchronous Tag Check Fault.	When FEAT_MTE2 is implemented
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	When FEAT_RAS is not implemented
0b100001	Alignment fault.	

0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented
0b110100	IMPLEMENTATION DEFINED fault (Lockdown).	
0b110101	IMPLEMENTATION DEFINED fault (Unsupported Exclusive or Atomic access).	

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The lookup level associated with MMU faults'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a trapped floating-point exception

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0	TFV	RES0												VECITR			IDF	RES0		IXF	UFF	OFF	DZF	IOF

Bit [24]

Reserved, RES0.

TFV, bit [23]

Trapped Fault Valid bit. Indicates whether the IDF, IXF, UFF, OFF, DZF, and IOF bits hold valid information about trapped floating-point exceptions.

TFV	Meaning
0b0	The IDF, IXF, UFF, OFF, DZF, and IOF bits do not hold valid information about trapped floating-point exceptions and are UNKNOWN.
0b1	One or more floating-point exceptions occurred during an operation performed while executing the reported instruction. The IDF, IXF, UFF, OFF, DZF, and IOF bits indicate trapped floating-point exceptions that occurred. For more information, see 'Floating-point exceptions and exception traps'.

It is IMPLEMENTATION DEFINED whether this field is set to 0 on an exception generated by a trapped floating-point exception from an instruction that is performing floating-point operations on more than one lane of a vector.

Note

This is not a requirement. Implementations can set this field to 1 on a trapped floating-point exception from an instruction and return valid information in the {IDF, IXF, UFF, OFF, DZF, IOF} fields.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [22:11]

Reserved, RES0.

VECITR, bits [10:8]

For a trapped floating-point exception from an instruction executed in AArch32 state this field is RES1.

For a trapped floating-point exception from an instruction executed in AArch64 state this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IDF	Meaning
0b0	Input denormal floating-point exception has not occurred.
0b1	Input denormal floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IXF	Meaning
0b0	Inexact floating-point exception has not occurred.
0b1	Inexact floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

UFF	Meaning
0b0	Underflow floating-point exception has not occurred.
0b1	Underflow floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

OFF	Meaning
0b0	Overflow floating-point exception has not occurred.
0b1	Overflow floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

DZF	Meaning
0b0	Divide by Zero floating-point exception has not occurred.
0b1	Divide by Zero floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IOF	Meaning
0b0	Invalid Operation floating-point exception has not occurred.
0b1	Invalid Operation floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

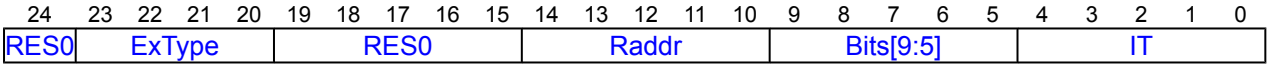
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a trapped floating-point exception

In an implementation that supports the trapping of floating-point exceptions:

- From an Exception level using AArch64, the FPCR.{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.
- From an Exception level using AArch32, the FPSCR.{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.

ISS encoding for a GCS exception



Bit [24]

Reserved, RES0.

ExType, bits [23:20]

The first level classification of GCS exceptions.

ExType	Meaning
0b0000	The exception reported is a Guarded Control Stack Data Check Exception.
0b0001	The exception reported is an EXLOCK Exception.
0b0010	The exception reported is a trap exception on GCSSTR or GCSSTTR instruction execution.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:15]

Reserved, RES0.

Raddr, bits [14:10]

When ExType == 0b0010 :

Indicates the data address register number supplied in the instruction that has been trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits[9:5]

When ExType == 0b0000 :

Rn, bits [4:0] of bits [9:5]

Indicates a register number used by the instruction that caused the Guarded Control Stack Data Check Exception.

For a procedure return instruction reported with ESR_EL2.ISS.IT as 0b00000, contains the register number for the register which contains the target address of the branch.

For a GCSPOPM instruction reported with ESR_EL2.ISS.IT as 0b00001, contains the register number for the register which is the destination register of the instruction.

For a procedure return instruction reported with ESR_EL2.ISS.IT as `0b00010` or `0b00011`, contains the value `0b11110`, indicating X30.

For a GCSSS1 instruction reported with ESR_EL2.ISS.IT as `0b00100`, contains the register number for the register which is the input register of the instruction.

If ESR_EL2.ISS.IT is reported as `0b00101` or `0b01000`, this field is UNKNOWN

If ESR_EL2.ISS.IT is reported as `0b01001`, this field is `0b11111`

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ExType == 0b0010 :

Rvalue, bits [4:0] of bits [9:5]

Indicates the data value register number supplied in the instruction that has been trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IT, bits [4:0]
When ExType == 0b0000 :

Type of the instruction that caused the Guarded Control Stack Data Check Exception.

IT	Meaning
`0b00000`	Guarded Control Stack Data Check Exception is from a procedure return instruction without Pointer authentication.
`0b00001`	Guarded Control Stack Data Check Exception is from a GCSPOPM instruction.
`0b00010`	Guarded Control Stack Data Check Exception is from a procedure return instruction with Pointer authentication that uses key A.
`0b00011`	Guarded Control Stack Data Check Exception is from a procedure return instruction with Pointer authentication that uses key B.
`0b00100`	Guarded Control Stack Data Check Exception is from a GCSSS1 instruction.
`0b00101`	Guarded Control Stack Data Check Exception is from a GCSSS2 instruction.
`0b01000`	Guarded Control Stack Data Check Exception is from a GCSPOPCX instruction.
`0b01001`	Guarded Control Stack Data Check Exception is from a GCSPOPX instruction.

All other values are reserved

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

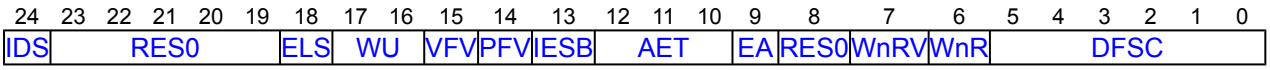
Reserved, RES0.

Additional information for the ISS encoding for a GCS exception

The following fields describe the configuration settings for the traps that are reported using EC value `0b101101` and ExType value `0b0010`:

- GCSCRE0_EL1.STREn
- GCSCR_EL1.STREn.
- GCSCR_EL2.STREn.
- GCSCR_EL3.STREn.
- HFGITR_EL2.nGCSSTR_EL1.

ISS encoding for an SError exception



Note

In earlier versions of the architecture, an SError exception is referred to as an SError interrupt or an asynchronous External abort exception.

IDS, bit [24]

IMPLEMENTATION DEFINED syndrome.

IDS	Meaning
`0b0`	Bits [23:0] of the ISS field holds the fields described in this encoding. **Note** If FEAT_RAS is not implemented, bits [23:0] of the ISS field are RES0.
`0b1`	Bits [23:0] of the ISS field holds IMPLEMENTATION DEFINED syndrome information that can be used to provide additional information about the SError exception.

Note

This field was previously called ISV.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:19]

Reserved, RES0.

ELS, bit [18]

When FEAT_RASv2 is implemented and DFSC == 0b010001:

Meaning of ELR_ELx.

ELS	Meaning
`0b0`	Asynchronous. Does not indicate the trigger for the exception.
`0b1`	Synchronous. The exception was triggered by the instruction at ELR_ELx.

SError exceptions that report this field is 1 are not required to be precise.

The ESR_EL2.AET field describes whether the exception is precise or imprecise.

Corrected, Recoverable or Restartable exceptions are precise. Unrecoverable or Uncontainable exceptions are imprecise.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WU, bits [17:16]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

Write Update. Describes whether a store instruction that generated an External abort updated the location.

WU	Meaning
0b00	Not a store instruction or translation table update, or the location might have been updated.
0b10	Store instruction or translation table update that did not update the location.
0b11	Store instruction or translation table update that updated the location.

In the description of this field, a store instruction is any memory-writing instruction that explicitly performs a store. This includes instructions that both read and write memory.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VFV, bit [15]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

FAR Valid. Indicates the FAR_EL2 register contains a valid virtual address.

VFV	Meaning
0b0	FAR_EL2 is not valid, and holds an UNKNOWN value.
0b1	FAR_EL2 contains a valid virtual address associated with the error.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PFV, bit [14]
When FEAT_PFAR is implemented and DFSC == 0b010001:

PFAR Valid. Describes whether the PFAR_EL2 register is valid.

PFV	Meaning
0b0	PFAR_EL2 is UNKNOWN.
0b1	PFAR_EL2 is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IESB, bit [13]
When FEAT_IESB is implemented and DFSC == 0b010001:

Implicit error synchronization event.

IESB	Meaning
0b0	The SError exception was either not synchronized by the implicit error synchronization event or not taken immediately.
0b1	The SError exception was synchronized by the implicit error synchronization event and taken immediately.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AET, bits [12:10]
When FEAT_RAS is implemented and DFSC == 0b010001:

Asynchronous Error Type.

Describes the PE error state after taking the SError exception.

AET	Meaning
0b000	Uncontainable (UC).
0b001	Unrecoverable state (UEU).
0b010	Restartable state (UEO).
0b011	Recoverable state (UER).
0b110	Corrected (CE).

All other values are reserved.

If multiple errors are taken as a single SError exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA, bit [9]
When FEAT_RAS is implemented and DFSC == 0b010001:

External abort type. Provides an IMPLEMENTATION DEFINED classification of External aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

WnRV, bit [7]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

ESR_EL2.WnR valid.

WnRV	Meaning
0b0	ESR_EL2.WnR is not valid and has been set to 0b0.
0b1	ESR_EL2.WnR is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WnR, bit [6]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

Write-not-Read. When the WnRV field is 0b1, indicates whether an exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

WnR	Meaning
0b0	Exception was caused by an instruction reading from a memory location.
0b1	Exception was caused by an instruction writing to a memory location.

Accessing this bit has the following behavior:

- This bit is RES0 if ESR_EL2.WnRV==0b0.
- This bit is not valid and reads UNKNOWN if an External abort on a Atomic access, reported with ESR_EL2.WU == 0b00.
- Otherwise RW.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DFSC, bits [5:0]

When FEAT_RAS is implemented:

Data Fault Status Code.

DFSC	Meaning
0b000000	Uncategorized error.
0b010001	Asynchronous SError exception.

All other values are reserved.

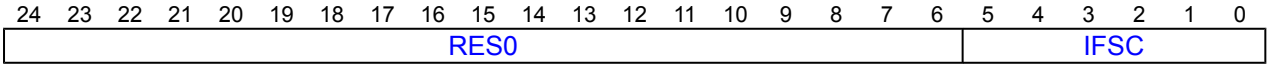
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ISS encoding for an exception from a Breakpoint or Vector Catch debug exception



Bits [24:6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

IFSC	Meaning
0b100010	Debug exception.

The reset behavior of this field is:

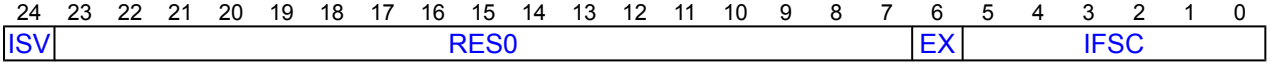
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a Breakpoint or Vector Catch debug exception

For more information about generating these exceptions:

- For exceptions from AArch64, see 'Breakpoint exceptions'.
- For exceptions from AArch32, see 'Breakpoint exceptions' and 'Vector Catch exceptions'.

ISS encoding for an exception from a Software Step exception



ISV, bit [24]

Instruction syndrome valid. Indicates whether the EX bit, ISS[6], is valid, as follows:

ISV	Meaning
0b0	EX bit is RES0.
0b1	EX bit is valid.

See the EX bit description for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:7]

Reserved, RES0.

EX, bit [6]

Exclusive operation. If the ISV bit is set to 1, this bit indicates whether a Load-Exclusive instruction was stepped.

EX	Meaning
0b0	An instruction other than a Load-Exclusive instruction was stepped.
0b1	A Load-Exclusive instruction was stepped.

If the ISV bit is set to 0, this bit is RES0, indicating no syndrome data is available.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IFSC, bits [5:0]

Instruction Fault Status Code.

IFSC	Meaning
0b100010	Debug exception.

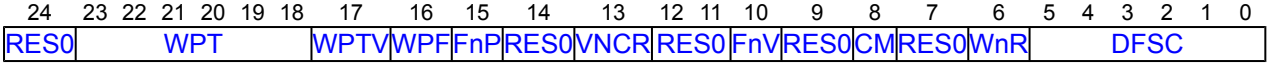
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a Software Step exception

For more information about generating these exceptions, see 'Software Step exceptions'.

ISS encoding for an exception from a Watchpoint exception



Bit [24]

Reserved, RES0.

WPT, bits [23:18]

When FEAT_Debugv8p2 is implemented:

Watchpoint number.

All other values are reserved.

Otherwise:

Reserved, RES0.

WPTV, bit [17]

When FEAT_Debugv8p2 is implemented:

Watchpoint number Valid.

WPTV	Meaning
0b0	The WPT field is invalid, and holds an UNKNOWN value.
0b1	The WPT field is valid, and holds the number of a watchpoint that triggered a Watchpoint exception.

If FEAT_Debugv8p9 is implemented, value 0b0 is not permitted.

When a Watchpoint exception is triggered by a watchpoint match:

- If FEAT_Debugv8p9 is implemented or the PE sets any of FnV, FnP, or WPF to 1, then the PE sets WPTV to 1.
- Otherwise, the PE sets WPTV to an IMPLEMENTATION DEFINED value, 0 or 1.

Otherwise:

Reserved, RES0.

WPF, bit [16]

Watchpoint might be false-positive.

WPF	Meaning	Applies when
0b0	The watchpoint matched an address or address range that was accessed by the instruction.	
0b1	The watchpoint matched an address or address range that might not have been accessed by the instruction.	When FEAT_SVE is implemented or FEAT_SME is implemented

Arm strongly recommends that this bit is set to 0, other than when one of the following instructions might generate a watchpoint match for an address or address range that the instruction does not access:

- An SVE contiguous vector load/store instruction, when the PE is in Streaming SVE mode.
- An SME load/store instruction.

FnP, bit [15]

FAR not Precise.

This field only has meaning if the FAR is valid; that is, when the FnV field is 0. If the FnV field is 1, the FnP field is 0.

FnP	Meaning	Applies when
0b0	If the FnV field is 0, the FAR holds the virtual address of an access or set of contiguous accesses that triggered a Watchpoint exception.	
0b1	The FAR holds any address within the smallest implemented translation granule that contains the virtual address of an access or set of contiguous accesses that triggered a Watchpoint exception.	When FEAT_SVE is implemented or FEAT_SME is implemented

Bit [14]

Reserved, RES0.

VNCR, bit [13]

Indicates that the watchpoint came from use of VNCR_EL2 register by EL1 code.

VNCR	Meaning	Applies when
0b0	The watchpoint was not generated by the use of VNCR_EL2 by EL1 code.	
0b1	The watchpoint was generated by the use of VNCR_EL2 by EL1 code.	When FEAT_NV2 is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [12:11]

Reserved, RES0.

FnV, bit [10]

FAR not Valid.

FnV	Meaning	Applies when
0b0	The FAR is valid, and its value is as described by the FnP field.	
0b1	The FAR is invalid, and holds an UNKNOWN value.	When FEAT_SVE is implemented or FEAT_SME is implemented

Bit [9]

Reserved, RES0.

CM, bit [8]

Cache maintenance. Indicates whether the Watchpoint exception came from a cache maintenance instruction:

CM	Meaning
0b0	The Watchpoint exception was not generated by the execution of one of the System instructions identified in the description of value 1.
0b1	The Watchpoint exception was generated by the execution of a cache maintenance instruction. The DC ZVA, DC GVA, and DC GZVA instructions are not classified as a cache maintenance instructions, and therefore their execution does not cause this field to be set to 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

WnR, bit [6]

Write not Read. Indicates whether the Watchpoint exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

WnR	Meaning
0b0	Watchpoint exception caused by an instruction reading from a memory location.
0b1	Watchpoint exception caused by an instruction writing to a memory location.

For Watchpoint exceptions on cache maintenance instructions, this bit always returns a value of 1.

For Watchpoint exceptions from an atomic instruction, this field is set to 0 if a read of the location would have generated the Watchpoint exception, otherwise it is set to 1.

If multiple watchpoints match on the same access, it is UNPREDICTABLE which watchpoint generates the Watchpoint exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

DFSC	Meaning
0b100010	Debug exception.

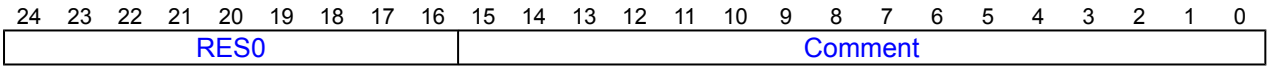
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a Watchpoint exception

For more information about generating these exceptions, see 'Watchpoint exceptions'.

ISS encoding for an exception from execution of a Breakpoint instruction



Bits [24:16]

Reserved, RES0.

Comment, bits [15:0]

Set to the instruction comment field value, zero extended as necessary.

For the AArch32 BKPT instructions, the comment field is described as the immediate field.

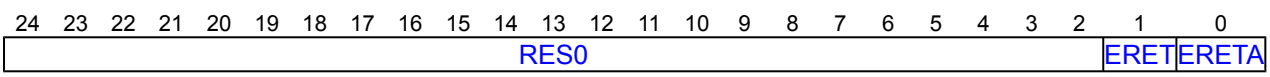
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from execution of a Breakpoint instruction

For more information about generating these exceptions, see 'Breakpoint instruction exceptions'.

ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction



This EC value applies when:

- FEAT_FGT is implemented.
- The Effective value of HCR_EL2.NV is 1.

Bits [24:2]

Reserved, RES0.

ERET, bit [1]

Indicates whether an ERET or ERETA* instruction was trapped to EL2.

ERET	Meaning
0b0	ERET instruction trapped to EL2.
0b1	ERETAA or ERETAB instruction trapped to EL2.

If this bit is 0, the ERETA field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ERETA, bit [0]

Indicates whether an ERETAA or ERETAB instruction was trapped to EL2.

ERETA	Meaning
0b0	ERETAA instruction trapped to EL2.
0b1	ERETAB instruction trapped to EL2.

When the ERET field is 0, this bit is RES0.

The reset behavior of this field is:

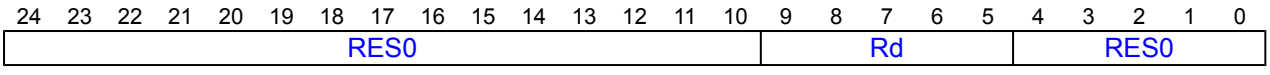
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction

For more information about generating these exceptions, see HCR_EL2.NV.

If FEAT_FGT is implemented, HFGITR_EL2.ERET controls fine-grained trap exceptions from ERET, ERETAA, and ERETAB execution.

ISS encoding for an exception from a TSTART instruction



Bits [24:10]

Reserved, RES0.

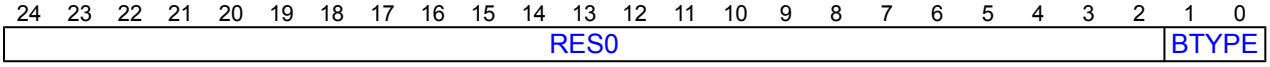
Rd, bits [9:5]

The Rd value from the issued instruction, the general purpose register used for the destination.

Bits [4:0]

Reserved, RES0.

ISS encoding for an exception from Branch Target Identification instruction



Bits [24:2]

Reserved, RES0.

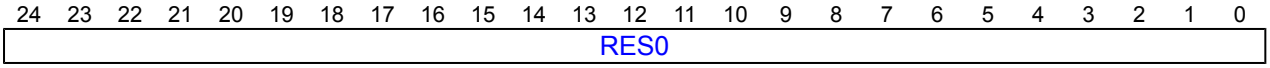
BTYPE, bits [1:0]

This field is set to the PSTATE.BTYPE value that generated the Branch Target Exception.

Additional information for the ISS encoding for an exception from Branch Target Identification instruction

For more information about generating these exceptions, see 'The AArch64 application level programmers model'.

ISS encoding for an exception from a trapped Pointer Authentication instruction



Bits [24:0]

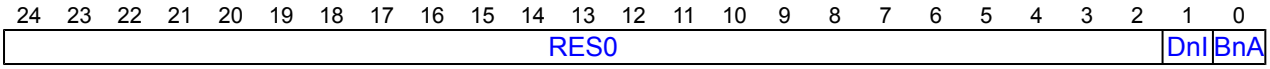
Reserved, RES0.

Additional information for the ISS encoding for an exception from a trapped Pointer Authentication instruction

For more information about generating these exceptions, see:

- HCR_EL2.API, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL2.
- SCR_EL3.API, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL3.

ISS encoding for a PAC Fail exception



Bits [24:2]

Reserved, RES0.

DnI, bit [1]

This field indicates whether the exception is as a result of an Instruction key or a Data key.

DnI	Meaning
0b0	Instruction Key.
0b1	Data Key.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BnA, bit [0]

This field indicates whether the exception is as a result of an A key or a B key.

BnA	Meaning
0b0	A key.
0b1	B key.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for a PAC Fail exception

The following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:

- AUTDA, AUTDZA.
- AUTDB, AUTDZB.
- AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIZA.
- AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZB.
- AUTIASPPC, AUTIASPPCR, AUTIA171615.
- AUTIBSPPC, AUTIBSPPCR, AUTIB171615.

If FEAT_FPACCOMBINE is implemented, the following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:

- RETAA, RETAB.
- RETAASPPC, RETABSPPC.
- RETAASPPCR, RETABSPPCR.
- BLRAA, BLRAAZ, BLRAB, BLRABZ.
- BRAA, BRAB, BRAAZ, BRABZ.
- ERETAA, ERETAB.
- LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

Accessing ESR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name ESR_EL2 or ESR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ESR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = ESR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ESR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ESR_EL2;
```

```
MSR ESR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        ESR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ESR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ESR_EL2 = X[t, 64];
```

```
MRS <Xt>, ESR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ESR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x138];
    else
        X[t, 64] = ESR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = ESR_EL2;
    else
        X[t, 64] = ESR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ESR_EL1;
```

```
MSR ESR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ESR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x138] = X[t, 64];
    else
        ESR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        ESR_EL2 = X[t, 64];
    else
        ESR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ESR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ESR_EL3, Exception Syndrome Register (EL3)

The ESR_EL3 characteristics are:

Purpose

Holds syndrome information for an exception taken to EL3.

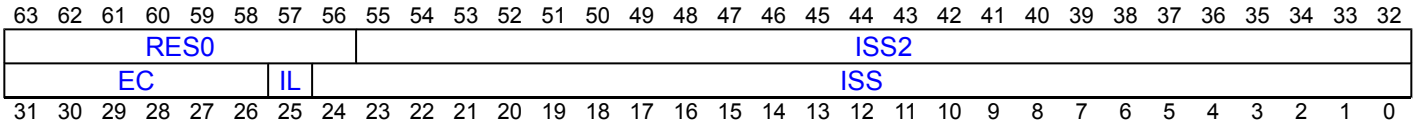
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ESR_EL3 are UNDEFINED.

Attributes

ESR_EL3 is a 64-bit register.

Field descriptions



ESR_EL3 is made UNKNOWN as a result of an exception return from EL3.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL3, the value of ESR_EL3 is UNKNOWN. The value written to ESR_EL3 must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

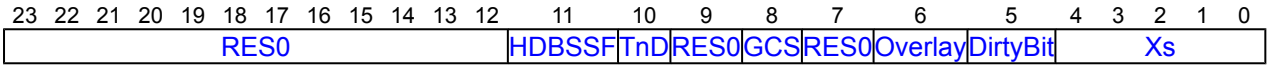
Bits [63:56]

Reserved, RES0.

ISS2, bits [55:32]

ISS2 encoding for an exception, the bit assignments are:

ISS2 encoding for an exception from a Data Abort



Bits [23:12]

Reserved, RES0.

HDBSSF, bit [11]

When FEAT_HDBSS is implemented, IsSecondStage(Fault), and DFSC == 0b0011xx || DFSC == 0b01001x || DFSC == 0b0101xx || DFSC == 0b10001x || DFSC == 0b1001xx:

Indicates that the fault was caused by the HDBSS.

When DFSC indicates a synchronous External abort on translation table walk or hardware update of translation table, this field indicates whether the fault was caused by a write to the HDBSS.

HDBSSF	Meaning
0b0	Fault was not caused by HDBSS.
0b1	Fault was caused by HDBSS.

This field only applies for stage 2 Permission faults.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TnD, bit [10]
When FEAT_MTE_CANONICAL_TAGS is implemented:

Tag not Data.

If a memory access generates a Data Abort for a stage 1 Permission fault, this field indicates whether the fault is due to an Allocation Tag access.

TnD	Meaning
0b0	Permission fault is not due to a write of an Allocation Tag to Canonically Tagged memory.
0b1	Permission fault is due to a write of an Allocation Tag to Canonically Tagged memory.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [9]

Reserved, RES0.

GCS, bit [8]
When FEAT_GCS is implemented:

Guarded Control Stack data access.

If a memory access generates a Data Abort, this field indicates whether the fault is due to a Guarded Control Stack data access.

GCS	Meaning
0b0	The Data Abort is not due to a Guarded control stack data access.
0b1	The Data Abort is due to a Guarded control stack data access. The ISV field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

Overlay, bit [6]
When FEAT_S1POE is implemented:

Overlay flag.

If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]
When FEAT_S1PIE is implemented:

DirtyBit flag.

If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

For any other fault or Access, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Xs, bits [4:0]
When FEAT_LS64 is implemented:

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort exception for a Translation fault, Access flag fault, or Permission fault, then this field holds register specifier, Xs.

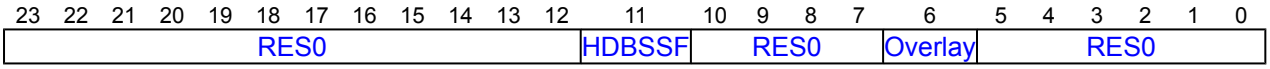
When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort exception for a Translation fault, Access flag fault, or Permission fault, then this field holds register specifier, Xs.

Otherwise, this field is RES0.

Otherwise:

Reserved, RES0.

ISS2 encoding for an exception from an Instruction Abort



Bits [23:12]

Reserved, RES0.

HDBSSF, bit [11]
When FEAT_HDBSS is implemented:

Indicates that the fault was caused by the HDBSS.

When IFSC indicates a synchronous External abort on translation table walk or hardware update of translation table, this field indicates whether the fault was caused by a write to the HDBSS.

HDBSSF	Meaning
0b0	Fault was not caused by HDBSS.
0b1	Fault was caused by HDBSS.

This field only applies for stage 2 Permission faults.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [10:7]

Reserved, RES0.

Overlay, bit [6]
When FEAT_S1POE is implemented:

Overlay flag.

If a memory access generates a Instruction Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Instruction Abort is not due to Overlay Permissions.
0b1	Instruction Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

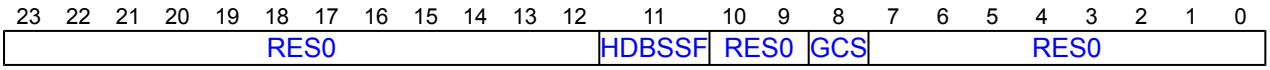
Otherwise:

Reserved, RES0.

Bits [5:0]

Reserved, RES0.

ISS2 encoding for a Granule Protection Check exception



Bits [23:12]

Reserved, RES0.

HDBSSF, bit [11]

When FEAT_HDBSS is implemented and IsSecondStage(Fault):

Indicates that the fault was caused by the HDBSS.

HDBSSF	Meaning
0b0	Fault was not caused by HDBSS.
0b1	Fault was caused by HDBSS.

This field only applies for stage 2 Permission faults.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [10:9]

Reserved, RES0.

GCS, bit [8]

When FEAT_GCS is implemented:

Guarded control stack data access.

Indicates that the Granule Protection Check Exception is due to a Guarded control stack data access.

GCS	Meaning
0b0	The Granule Protection Check Exception is not due to a Guarded control stack data access.
0b1	The Granule Protection Check Exception is due to a Guarded control stack data access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

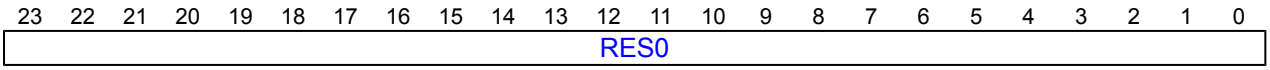
Otherwise:

Reserved, RES0.

Bits [7:0]

Reserved, RES0.

ISS2 encoding for all other exceptions



Bits [23:0]

Reserved, RES0.

EC, bits [31:26]

Exception Class. Indicates the reason for the exception that this register holds information about.

For each EC value, the table references a subsection that gives information about:

- The cause of the exception, for example the configuration required to enable the trap.
- The encoding of the associated ISS.

Possible values of the EC field are:

EC	Meaning	ISS	ISS2	Applies when
0b000000	Unknown reason.	ISS encoding for exceptions with an unknown reason	ISS2 encoding for all other exceptions	
0b000001	Trapped WF* instruction execution. Conditional WF* instructions that fail their condition code check do not cause an exception.	ISS encoding for an exception from a WF* instruction	ISS2 encoding for all other exceptions	
0b000011	Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC value 0b000000.	ISS encoding for an exception from an MCR or MRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000100	Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC value 0b000000.	ISS encoding for an exception from an MCRR or MRRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000101	Trapped MCR or MRC access with (coproc==0b1110).	ISS encoding for an exception from an MCR or MRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000110	Trapped LDC or STC access. The only architected uses of these instruction are: • An STC to write data to memory from DBGDTRRXint. • An LDC to read data from memory to DBGDTRTXint.	ISS encoding for an exception from an LDC or STC instruction	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b000111	Access to SME, SVE, Advanced SIMD or floating-point functionality trapped by CPACR_EL1.FPEN, CPTR_EL2.FPEN, CPTR_EL2.TFP, or CPTR_EL3.TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2.TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000.	ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps	ISS2 encoding for all other exceptions	
0b001001	Trapped use of a Pointer authentication instruction.	ISS encoding for an exception from a trapped Pointer Authentication instruction	ISS2 encoding for all other exceptions	When FEAT_PAuth is implemented
0b001010	Trapped execution of any instruction not covered by other EC values.	ISS encoding for an exception from any other instruction	ISS2 encoding for all other exceptions	When FEAT_LS64 is implemented
0b001100	Trapped MRRC access with (coproc==0b1110).	ISS encoding for an exception from an MCRR or MRRC access	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented

0b001101	Branch Target Exception.	ISS encoding for an exception from Branch Target Identification instruction	ISS2 encoding for all other exceptions	When FEAT_BTI is implemented
0b001110	Illegal Execution state.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault	ISS2 encoding for all other exceptions	
0b010011	SMC instruction execution in AArch32 state, when SMC is not disabled.	ISS encoding for an exception from SMC instruction execution in AArch32 state	ISS2 encoding for all other exceptions	When FEAT_AA32 is implemented
0b010100	Trapped MSRR, MRRS or System instruction execution in AArch64 state, that is not reported using EC value 0b000000.	ISS encoding for an exception from MSRR, MRRS, or 128-bit System instruction execution in AArch64 state	ISS2 encoding for all other exceptions	When FEAT_SYSREG128 is implemented or FEAT_SYSINSTR128 is implemented
0b010101	SVC instruction execution in AArch64 state.	ISS encoding for an exception from HVC or SVC instruction execution	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b010110	HVC instruction execution in AArch64 state, when HVC is not disabled.	ISS encoding for an exception from HVC or SVC instruction execution	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b010111	SMC instruction execution in AArch64 state, when SMC is not disabled.	ISS encoding for an exception from SMC instruction execution in AArch64 state	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b011000	Trapped MSR, MRS or System instruction execution in AArch64 state, that is not reported using EC values 0b000000, 0b000001 or 0b000111. This includes all instructions that cause exceptions that are part of the encoding space defined in 'System instruction class encoding overview', except for those exceptions reported using EC values 0b000000, 0b000001, or 0b000111.	ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b011001	Access to SVE functionality trapped as a result of CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ, that is not reported using EC value 0b000000.	ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ	ISS2 encoding for all other exceptions	When FEAT_SVE is implemented
0b011011	Exception from an access to a TSTART instruction at EL0 when SCTLR_EL1.TME0 == 0, EL0 when SCTLR_EL2.TME0 == 0, at EL1 when	ISS encoding for an exception from a TSTART instruction	ISS2 encoding for all other exceptions	When FEAT_TME is implemented

	SCTLR_EL1.TME == 0, at EL2 when SCTLR_EL2.TME == 0 or at EL3 when SCTLR_EL3.TME == 0.			
0b011100	Exception from a PAC Fail	ISS encoding for a PAC Fail exception	ISS2 encoding for all other exceptions	When FEAT_FPAC is implemented
0b011101	Access to SME functionality trapped as a result of CPACR_EL1.SMEN, CPTR_EL2.SMEN, CPTR_EL2.TSM, CPTR_EL3.ESM, or an attempted execution of an instruction that is illegal because of the value of PSTATE.SM or PSTATE.ZA, that is not reported using EC value 0b000000.	ISS encoding for an exception due to SME functionality	ISS2 encoding for all other exceptions	When FEAT_SME is implemented
0b011110	Granule Protection Check exception	ISS encoding for a Granule Protection Check exception	ISS2 encoding for a Granule Protection Check exception	When FEAT_RME is implemented
0b011111	IMPLEMENTATION DEFINED exception to EL3.	ISS encoding for an IMPLEMENTATION DEFINED exception to EL3	ISS2 encoding for all other exceptions	
0b100000	Instruction Abort from a lower Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.	ISS encoding for an exception from an Instruction Abort	ISS2 encoding for an exception from an Instruction Abort	
0b100001	Instruction Abort taken without a change in Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.	ISS encoding for an exception from an Instruction Abort	ISS2 encoding for all other exceptions	
0b100010	PC alignment fault exception.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault	ISS2 encoding for all other exceptions	
0b100100	Data Abort exception from a lower Exception level. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including	ISS encoding for an exception from a Data Abort	ISS2 encoding for an exception from a Data Abort	

	synchronous parity or ECC errors. Not used for debug-related exceptions.			
0b100101	Data Abort exception taken without a change in Exception level. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.	ISS encoding for an exception from a Data Abort	ISS2 encoding for an exception from a Data Abort	
0b100110	SP alignment fault exception.	ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault	ISS2 encoding for all other exceptions	
0b100111	Memory Operation Exception.	ISS encoding for an exception from the Memory Copy and Memory Set instructions	ISS2 encoding for all other exceptions	When FEAT_MOPS is implemented
0b101100	Trapped floating-point exception taken from AArch64 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.	ISS encoding for an exception from a trapped floating-point exception	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b101101	GCS exception.	ISS encoding for a GCS exception	ISS2 encoding for all other exceptions	When FEAT_GCS is implemented
0b101111	SError exception.	ISS encoding for an SError exception	ISS2 encoding for all other exceptions	
0b111100	BRK instruction execution in AArch64 state. This is reported in ESR_EL3 only if a BRK instruction is executed in EL3. This is the only debug exception that can be taken to EL3 when EL3 is using AArch64.	ISS encoding for an exception from execution of a Breakpoint instruction	ISS2 encoding for all other exceptions	When FEAT_AA64 is implemented
0b111101	Profiling exception	ISS encoding for a Profiling exception	ISS2 encoding for all other exceptions	When FEAT_EBEP is implemented, or FEAT_SPE_EXC is implemented, or FEAT_TRBE_EXC is implemented

All other EC values are reserved by Arm, and:

- Unused values in the range 0b000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.

- Unused values in the range `0b101101` - `0b111111` (`0x2D` - `0x3F`) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [25]

Instruction Length for synchronous exceptions. Possible values of this bit are:

IL	Meaning
`0b0`	16-bit instruction trapped.
`0b1`	32-bit instruction trapped. This value is also used when the exception is one of the following: • An SError exception. • An Instruction Abort exception. • A PC alignment fault exception. • An SP alignment fault exception. • A Data Abort exception for which the value of the ISV bit is 0. • An Illegal Execution state exception. • Any debug exception except for Breakpoint instruction exceptions. • An exception reported using EC value `0b000000`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [24:0]

Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

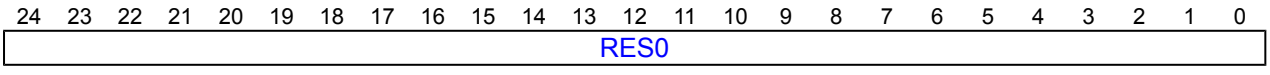
Typically, an ISS encoding has a number of subfields. When an ISS subfield holds a register number, the value returned in that field is the AArch64 view of the register number.

For an exception taken from AArch32 state, see 'Mapping of the general-purpose registers between the Execution states'.

If the AArch32 register descriptor is `0b1111`, then:

- If the instruction that generated the exception was not UNPREDICTABLE, the field takes the value `0b11111`.
- If the instruction that generated the exception was UNPREDICTABLE, the field takes an UNKNOWN value that must be either:
  - The AArch64 view of the register number of a register that might have been used at the Exception level from which the exception was taken.
  - The value `0b11111`.

ISS encoding for exceptions with an unknown reason



Bits [24:0]

Reserved, RES0.

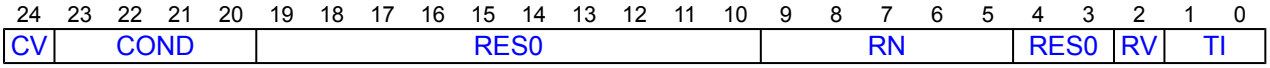
Additional information for the ISS encoding for exceptions with an unknown reason

When an exception is reported using this EC value, the IL field is set to 1.

This EC value is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or that is not accessible at the current Exception level and Security state, including:
    - A read access using a System register pattern that is not allocated for reads or that does not permit reads at the current Exception level and Security state.
    - A write access using a System register pattern that is not allocated for writes or that does not permit writes at the current Exception level and Security state.
    - Instruction encodings that are unallocated.
    - Instruction encodings for instructions or System registers that are not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
- In AArch32 state, attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the SCTLR_EL1.{ITD, SED, CP15BEN} control bits.
- Attempted execution of:
    - An HVC instruction when disabled by HCR_EL2.HCD or SCR_EL3.HCE.
    - An SMC instruction when disabled by SCR_EL3.SMD.
    - An HLT instruction when disabled by EDSCR.HDE.
- Attempted execution of an MSR or MRS instruction to access SP_EL0 when the value of SPSel.SP is 0.
- Attempted execution of an MSR or MRS instruction using a _EL12 register name when the Effective value of HCR_EL2.E2H is not 1.
- Attempted execution, in Debug state, of:
    - A DCPS1 instruction when the value of HCR_EL2.TGE is 1 and EL2 is disabled or not implemented in the current Security state.
    - A DCPS2 instruction from EL1 or EL0 when EL2 is disabled or not implemented in the current Security state.
    - A DCPS3 instruction when the value of EDSCR.SDD is 1, or when EL3 is not implemented.
- When EL3 is using AArch64, attempted execution from Secure EL1 of an SRS instruction using R13_mon.
- In Debug state when the value of EDSCR.SDD is 1, the attempted execution at EL2, EL1, and EL0 of an instruction that is configured to trap to EL3.
- In AArch32 state, the attempted execution of an MRS (banked register) or an MSR (banked register) instruction to SPSR_mon, SP_mon, or LR_mon.
- An exception that is taken to EL2 because the value of HCR_EL2.TGE is 1. If the value of HCR_EL2.TGE is 0, this exception is reported using an ESR_EL3.EC value of `0b000111`.
- In Non-transactional state, attempted execution of a TCOMMIT instruction.

ISS encoding for an exception from a WF* instruction



CV, bit [24]

Condition code valid.

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to `0b1110`.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:10]

Reserved, RES0.

RN, bits [9:5]

When FEAT_WFxT is implemented:

Register Number. Indicates the register number supplied for a `WFET` or `WFIT` instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

RV, bit [2]

When FEAT_WFxT is implemented:

Register field Valid.

If TI[1] == 1, then this field indicates whether RN holds a valid register number for the register argument to the trapped `WFET` or `WFIT` instruction.

RV	Meaning
`0b0`	Register field invalid.
`0b1`	Register field valid.

If TI[1] == 0, then this field is RES0.

This field is set to 1 on a trap on `WFET` or `WFIT`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TI, bits [1:0]

Trapped instruction. Possible values of this bit are:

TI	Meaning	Applies when
0b00	WFI trapped.	
0b01	WFE trapped.	
0b10	WFIT trapped.	When FEAT_WFxT is implemented
0b11	WFET trapped.	When FEAT_WFxT is implemented

When FEAT_WFxT is implemented, this is a two bit field as shown. Otherwise, bit[1] is RES0.

The reset behavior of this field is:

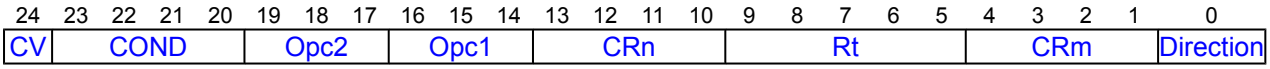
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a WF* instruction

The following fields describe configuration settings for generating this exception:

- HCR.{TWE, TWI}.
- SCTLR_EL1.{nTWE, nTWI}.
- SCTLR_EL2.{nTWE, nTWI}.
- HCR_EL2.{TWE, TWI}.
- SCR_EL3.{TWE, TWI}.

ISS encoding for an exception from an MCR or MRC access



CV, bit [24]

Condition code valid.

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc2, bits [19:17]

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value `0b000`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [16:14]

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value `0b111`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

If the Rt value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:

- The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
- The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value `0b0000`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
`0b0`	Write to System register space. MCR instruction.
`0b1`	Read from System register space. MRC or VMRS instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an MCR or MRC access

The following fields describe configuration settings for generating exceptions from an MCR or MRC access using coproc `0b1111`, that are reported using EC value `0b000011`:

- If FEAT_TIDCP1 is implemented, SCTLR_EL1.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch32 state, trapped to EL1.
- CNTKCTL_EL1.{EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- PMUSERENR_EL0.{ER, CR, SW, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- AMUSERENR_EL0.EN, for accesses to Activity Monitors registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- HCR.{TRVM, TVM} and HCR_EL2.{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, trapped to EL2.
- HCR.TTLB and HCR_EL2.TTLB, for execution of TLB maintenance instructions at EL1 using AArch32 state, trapped to EL2.
- HCR.{TSW, TPC, TPU} and HCR_EL2.{TSW, TPC, TPU} for execution of cache maintenance instructions at EL0 and EL1 using AArch32 state, trapped to EL2.
- HCR.TAC and HCR_EL2.TACR, for accesses to the Auxiliary Control Register at EL1 using AArch32 state, trapped to EL2.
- HCR.TIDCP and HCR_EL2.TIDCP, for accesses to lockdown, DMA, and TCM operations at EL0 and EL1 using AArch32 state, trapped to EL2.
- If FEAT_TIDCP1 is implemented, SCTLR_EL2.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch32 state, trapped to EL2.
- HCR.{TID1, TID2, TID3} and HCR_EL2.{TID1, TID2, TID3}, for accesses to ID registers at EL0 and EL1 using AArch32 state, trapped to EL2.
- HCR2.TERR, for Non-secure accesses to error record registers at EL1 using AArch32 state, trapped to EL2.
- HCPTR.TCPAC and CPTR_EL2.TCPAC, for accesses to CPACR_EL1 or CPACR using AArch32 state, trapped to EL2.
- HSTR.T<n> and HSTR_EL2.T<n>, for accesses to System registers using AArch32 state, trapped to EL2.
- CNTHCTL.PL1PCEN and CNTHCTL_EL2.EL1PCEN, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HDCR.TTRF, for Non-secure accesses to trace filter control registers from system registers using AArch32 state, trapped to EL2.
- HDCR.{TPM, TPMCR} and MDCR_EL2.{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HCPTR.TAM and CPTR_EL2.TAM, for accesses to Activity Monitors registers from EL0 and EL1 using AArch32 state, trapped to EL2.

- CPTR_EL3.TCPAC, for accesses to CPACR from EL1 and EL2, and accesses to HCPTR from EL2 using AArch32 state, trapped to EL3.
- MDCR_EL3.TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- CPTR_EL3.TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- If FEAT_FGT is implemented, access to some registers at EL0, trapped to EL2.

The following fields describe configuration settings for generating exceptions from an MCR or MRC access using coproc `0b1110`, that are reported using EC value `0b000101`:

- CPACR_EL1.TTA for accesses to trace registers, trapped to EL1 or EL2.
- MDSCR_EL1.TDCC, for accesses to the Debug Communications Channel (DCC) registers at EL0 and EL1 using AArch32 state, trapped to EL1 or EL2.
- If FEAT_FGT is implemented, MDCR_EL2.TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and MDCR_EL3.TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- HCR.TID0 and HCR_EL2.TID0, for accesses to the JIDR register in the ID group 0 at EL0 and EL1 using AArch32, trapped to EL2.
- HCPTR.TTA and CPTR_EL2.TTA, for accesses to trace registers using AArch32, trapped to EL2.
- HDCR.TDRA and MDCR_EL2.TDRA, for accesses to Debug ROM registers DBGDRAR and DBGDSAR using AArch32, trapped to EL2.
- HDCR.TDOSA and MDCR_EL2.TDOSA, for accesses to powerdown debug registers, using AArch32 state, trapped to EL2.
- HDCR.TDA and MDCR_EL2.TDA, for accesses to other debug registers, using AArch32 state, trapped to EL2.
- CPTR_EL3.TTA, for accesses to trace registers using AArch32, trapped to EL3.
- MDCR_EL3.TDOSA, for accesses to powerdown debug registers using AArch32, trapped to EL3.
- MDCR_EL3.TDA, for accesses to other debug registers, using AArch32, trapped to EL3.

The following fields describe configuration settings for generating exceptions from a VMSR or VMRS access, that are reported using EC value `0b001000`:

- HCR.TID0 and HCR_EL2.TID0, for accesses to the FPSID register in ID group 0 at EL1 using AArch32 state, VMRS access trapped to EL2.
- HCR.TID3 and HCR_EL2.TID3, for accesses to registers in ID group 3 including MVFR0, MVFR1 and MVFR2, VMRS access trapped to EL2.
- HCPTR.{TCP10, TCP11}, for Non-secure accesses to FPSCR, FPSID, FPEXC, MVFR0, MVFR1, and MVFR2, trapped to EL2.

ISS encoding for an exception from any other instruction

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ISS																								

ISS, bits [24:0]

ISS	Meaning	Applies when
`0b00000000000000000000000000`	`ST64BV` instruction trapped.	When FEAT_LS64_V is implemented
`0b00000000000000000000000001`	`ST64BV0` instruction trapped.	When FEAT_LS64_ACCDATA is implemented
`0b00000000000000000000000010`	`LD64B` or `ST64B` instruction trapped.	When FEAT_LS64 is implemented

All other values are reserved.

ISS encoding for an exception from an MCRR or MRRC access

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CV	COND				Opc1				RES0	Rt2					Rt					CRm				Direction

CV, bit [24]

Condition code valid.

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [19:16]

The Opc1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

Rt2, bits [14:10]

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer.

If the Rt2 value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt2 value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b11111.

- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
    - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
    - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer.

If the Rt value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
    - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
    - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
`0b0`	Write to System register space. MCRR instruction.
`0b1`	Read from System register space. MRRC instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an MCRR or MRRC access

The following fields describe configuration settings for generating exceptions from an MCRR or MRRC access using coproc `0b1111`, that are reported using EC value `0b000100`:

- CNTKCTL_EL1.{EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, trapped to EL1 or EL2.
- PMUSERENR_EL0.{CR, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, trapped to EL1 or EL2.

- AMUSERENR_EL0.{EN}, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 using AArch32 state, trapped to EL1 or EL2.
- HCR.{TRVM, TVM} and HCR_EL2.{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, trapped to EL2.
- HCR2.TERR, for Non-secure accesses to error record registers at EL1 using AArch32 state, trapped to EL2.
- HSTR.T<n> and HSTR_EL2.T<n>, for accesses to System registers using AArch32 state, trapped to EL2.
- CNTHCTL.{PL1PCEN, PL1PCTEN} and CNTHCTL_EL2.{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HDCR.TPM and MDCR_EL2.{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, trapped to EL2.
- HCPTR.TAM and CPTR_EL2.TAM, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 and EL1 using AArch32 state, trapped to EL2.
- MDCR_EL3.TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- CPTR_EL3.TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, trapped to EL3.
- If FEAT_FGT is implemented, HDFGRTR_EL2.PMCCNTR_EL0 for MRRC access and HDFGWTR_EL2.PMCCNTR_EL0 for MCRR access to PMCCNTR at EL0, trapped to EL2.

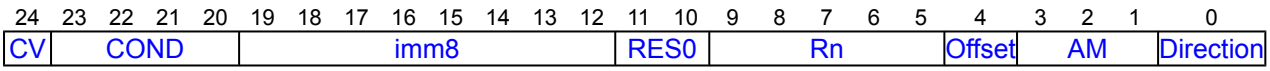
The following fields describe configuration settings for generating exceptions from an MCRR or MRRC access using coproc `0b1110`, that are reported using EC value `0b001100`:

- MDSCR_EL1.TDCC, for accesses to the Debug ROM registers DBGDSAR and DBGDRAR at EL0 using AArch32 state, trapped to EL1 or EL2.
- HDCR.TDRA and MDCR_EL2.TDRA, for accesses to Debug ROM registers DBGDRAR and DBGDSAR using AArch32, trapped to EL2.
- MDCR_EL3.TDA, for accesses to debug registers, using AArch32, trapped to EL3.
- CPACR_EL1.TTA for accesses to trace registers using AArch32, trapped to EL1 or EL2.
- HCPTR.TTA and CPTR_EL2.TTA, for accesses to trace registers using AArch32, trapped to EL2.
- CPTR_EL3.TTA, for accesses to trace registers using AArch32, trapped to EL3.

Note

If the Armv8-A architecture is implemented with an ETMv4 implementation, MCRR and MRRC accesses to trace registers are UNDEFINED and the resulting exception is higher priority than an exception due to these traps.

ISS encoding for an exception from an LDC or STC instruction



CV, bit [24]

Condition code valid.

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to `0b1110`.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

imm8, bits [19:12]

The immediate value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

Rn, bits [9:5]

The Rn value from the issued instruction, the general-purpose register used for the transfer.

If the Rn value is not `0b1111`, then the reported value gives the AArch64 view of the register. Otherwise, if the Rn value is `0b1111`:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value `0b11111`.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
  - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
  - The value `0b11111`.

See 'Mapping of the general-purpose registers between the Execution states'.

This field is valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction. When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Offset, bit [4]

Indicates whether the offset is added or subtracted:

Offset	Meaning
0b0	Subtract offset.
0b1	Add offset.

This bit corresponds to the U bit in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AM, bits [3:1]

Addressing mode. The permitted values of this field are:

AM	Meaning
0b000	Immediate unindexed.
0b001	Immediate post-indexed.
0b010	Immediate offset.
0b011	Immediate pre-indexed.
0b100	For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved.
0b110	For a trapped STC instruction, this encoding is reserved.

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in System and memory-mapped registers and translation table entries'.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write to memory. STC instruction.
0b1	Read from memory. LDC instruction.

The reset behavior of this field is:

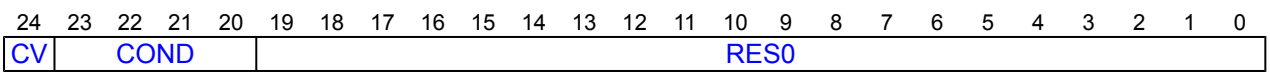
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from an LDC or STC instruction

The following fields describe the configuration settings from an LDC or STC access for the traps that are reported using EC value 0b000110:

- MDSCR_EL1.TDCC, for accesses to DBGDTRTXint and DBGDTRRXint, using AArch32 state, trapped to EL1 or EL2.
- HDCR.TDA and MDCR_EL2.TDA, for accesses to DBGDTRTXint and DBGDTRRXint, using AArch32 state, trapped to EL2.
- MDCR_EL3.TDA, for accesses to DBGDTRTXint and DBGDTRRXint, using AArch32 state, trapped to EL3.
- If FEAT_FGT is implemented, MDCR_EL2.TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and MDCR_EL3.TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.

ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps



The accesses covered by this trap include:

- Execution of SVE or Advanced SIMD and floating-point instructions.
- Accesses to the Advanced SIMD and floating-point System registers.
- Execution of SME instructions.

For an implementation that does not include either SVE or support for Advanced SIMD and floating-point, the exception is reported using the EC value `0b000000`.

CV, bit [24]

Condition code valid.

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to `0b1110`.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to `0b1110`.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to `0b1110`, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:0]

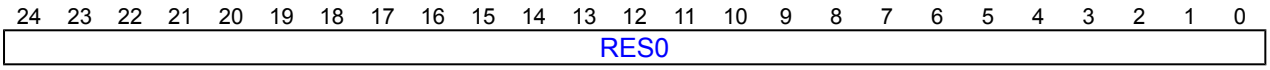
Reserved, RES0.

Additional information for the ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps

The following fields describe the configuration settings for the traps that are reported using EC value `0b000111`:

- HCPTR.{TCP10, TCP11}, for Non-secure accesses to Advanced SIMD and floating-point registers and instructions, trapped to EL2.
- HCPTR.TASE, for Non-secure accesses to Advanced SIMD functionality, trapped to EL2.
- CPACR_EL1.FPEN, for accesses to SIMD and floating-point registers trapped to EL1.
- CPTR_EL2.FPEN and CPTR_EL2.TFP, for accesses to SIMD and floating-point registers trapped to EL2.
- CPTR_EL3.TFP, for accesses to SIMD and floating-point registers trapped to EL3.

ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ



The accesses covered by this trap include:

- Execution of SVE instructions when the PE is not in Streaming SVE mode.
- Accesses to the SVE System registers, ZCR_ELx.

For an implementation that does not include SVE, the exception is reported using the EC value `0b000000`.

Bits [24:0]

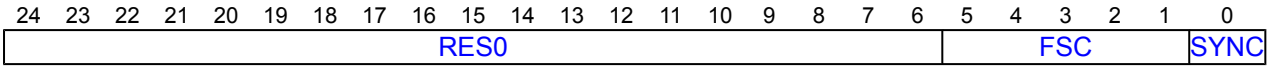
Reserved, RES0.

Additional information for the ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ

The following fields describe the configuration settings for the traps that are reported using EC value `0b011001`:

- CPACR_EL1.ZEN, for execution of SVE instructions and accesses to SVE registers at EL0 or EL1, trapped to EL1.
- CPTR_EL2.ZEN and CPTR_EL2.TZ, for execution of SVE instructions and accesses to SVE registers at EL0, EL1, or EL2, trapped to EL2.
- CPTR_EL3.EZ, for execution of SVE instructions and accesses to SVE registers from all Exception levels, trapped to EL3.

ISS encoding for a Profiling exception



Bits [24:6]

Reserved, RES0.

FSC, bits [5:1]

Indicates why the Profiling exception was generated.

FSC	Meaning	Applies when
0b00000	PMU Profiling exception. One of the following applies, and ESR_EL3.SYNC describes which: • The exception was generated because at least one PMU counter overflow status flag was 1, and was taken asynchronously. • The exception was generated because FEAT_SEBEP is implemented and PSTATE.PPEND was 1, and was taken synchronously.	When FEAT_EBEP is implemented
0b00001	Profiling Buffer management event. The exception was generated because PMBSR_EL3.S was 1.	When FEAT_SPE_EXC is implemented
0b00010	Trace buffer management event. The exception was generated because TRBSR_EL3.IRQ was 1.	When FEAT_TRBE_EXC is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SYNC, bit [0]

Indicates whether the Profiling exception was taken synchronously or asynchronously.

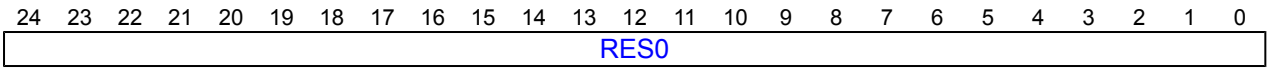
SYNC	Meaning	Applies when
0b0	The exception was taken asynchronously.	
0b1	The exception was taken synchronously.	When FEAT_SEBEP is implemented

If ESR_EL3.FSC does not indicate a PMU Profiling exception, or FEAT_SEBEP is not implemented, then the only permitted value is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault



Bits [24:0]

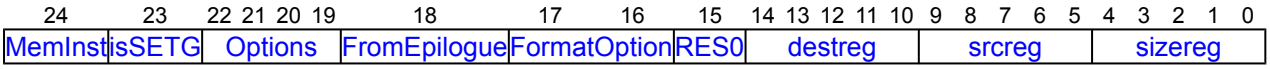
Reserved, RES0.

Additional information for the ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault

There are no configuration settings for generating Illegal Execution state exceptions and PC alignment fault exceptions. For more information about PC alignment fault exceptions, see 'PC alignment checking'.

'SP alignment checking' describes the configuration settings for generating SP alignment fault exceptions.

ISS encoding for an exception from the Memory Copy and Memory Set instructions



MemInst, bit [24]

Indicates the memory instruction class causing the exception.

MemInst	Meaning
0b0	CPYFE*, CPYFM*, CPYE*, and CPYM* instructions.
0b1	SETE*, SETM*, SETGE*, and SETGM* instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

isSETG, bit [23]

Indicates whether the instruction belongs to SETGM* or SETGE* class of instruction.

isSETG	Meaning
0b0	Not a SETGM* or SETGE* instruction.
0b1	SETGM* or SETGE* instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Options, bits [22:19]

Options : the Options field of the instruction.

For Memory Copy instructions, bits[22:19] forms the Options field, which holds the bits[15:12] of the instruction.

For Memory Set instructions:

- Bits[22:21] are RES0.
- Bits[20:19] form the Options field, which holds the bits[13:12] of the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FromEpilogue, bit [18]

Indicates whether the instruction belongs to the epilogue class of Memory Copy or Memory Set instructions.

FromEpilogue	Meaning
0b0	Not an epilogue instruction.
0b1	CPYE*, CPYFE*, SETE*, or SETGE* instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FormatOption, bits [17:16]

Reports the Option used to encode the initial Xs, Xd, and Xn register values provided to the instruction that generated the exception.

FormatOption	Meaning
0b00	Option B.
0b01	Option A.
0b10	Option A.
0b11	Option B.

For more information, see Memory Copy and Memory Set exceptions.

Note

This field was previously presented as two separate bits, WrongOption, bit[17] and OptionA, bit [16], which were already expected to be used together and not individually.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

destreg, bits [14:10]

The destination register value from the issued instruction, containing the destination address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

srcreg, bits [9:5]

The source register value from the issued instruction, containing either the source address or the source data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

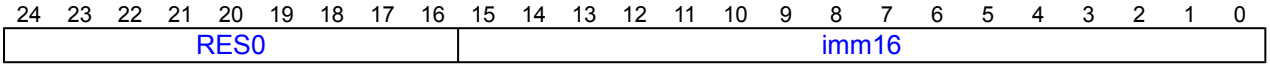
sizereg, bits [4:0]

The size register value from the issued instruction, containing the number of bytes to be transfered or set.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from HVC or SVC instruction execution



Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, and for an A64 SVC instruction, this is the value of the imm16 field of the issued instruction.

For an A32 or T32 SVC instruction:

- If the instruction is unconditional, then:
  - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
  - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from HVC or SVC instruction execution

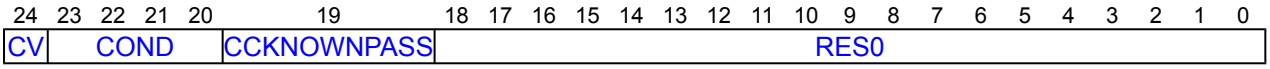
In AArch32 state, the HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.

For T32 and A32 instructions, see 'SVC' and 'HVC'.

For A64 instructions, see 'SVC' and 'HVC'.

If FEAT_FGT is implemented, HFGITR_EL2.{SVC_EL1, SVC_EL0} control fine-grained traps on SVC execution.

ISS encoding for an exception from SMC instruction execution in AArch32 state



For an SMC instruction that completes normally and generates an exception that is taken to EL3, the ISS encoding is RES0.

For an SMC instruction that is trapped to EL2 from EL1 because HCR_EL2.TSC is 1, the ISS encoding is as shown in the diagram.

CV, bit [24]

Condition code valid.

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:

        - With COND set to 0b1110, the value for unconditional.
        - With the COND value held in the instruction.
    - When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
        - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
        - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
    - For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CCKNOWNPASS, bit [19]

Indicates whether the instruction might have failed its condition code check.

CCKNOWNPASS	Meaning
0b0	The instruction was unconditional, or was conditional and passed its condition code check.
0b1	The instruction was conditional, and might have failed its condition code check.

Note

In an implementation in which an SMC instruction that fails it code check is not trapped, this field can always return the value 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [18:0]

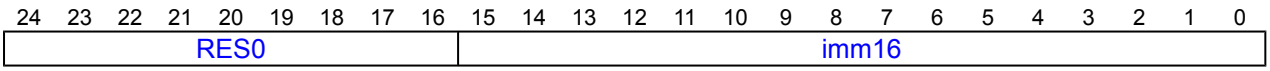
Reserved, RES0.

Additional information for the ISS encoding for an exception from SMC instruction execution in AArch32 state

HCR.TSC describes the configuration settings for trapping SMC instructions to EL2.

HCR_EL2.TSC describes the configuration settings for trapping SMC instructions to EL2.

ISS encoding for an exception from SMC instruction execution in AArch64 state



Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the issued SMC instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

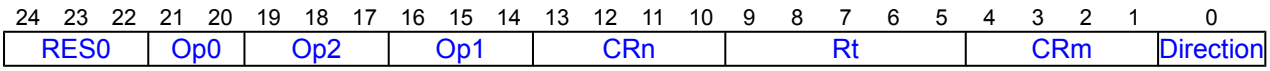
Additional information for the ISS encoding for an exception from SMC instruction execution in AArch64 state

The value of ISS[24:0] described here is used both:

- When an SMC instruction is trapped from EL1 modes.
- When an SMC instruction is not trapped, so completes normally and generates an exception that is taken to EL3.

HCR_EL2.TSC describes the configuration settings for trapping SMC from EL1 modes.

ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state



Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

For system instructions which require that the opcode Rt field is set to `0b11111`, but where the trapped instruction has a different value of Rt, an implementation is permitted to return the value `0b11111`, instead of the value of Rt from the trapped instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write access, including MSR instructions.
0b1	Read access, including MRS instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state

For exceptions caused by System instructions, see 'System instructions' subsection of 'Branches, exception generating and System instructions' for the encoding values returned by an instruction.

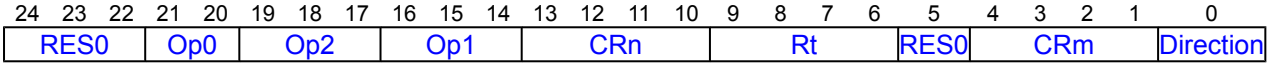
The following fields describe configuration settings for generating the exception that is reported using EC value 0b011000:

- If FEAT_TIDCP1 is implemented, SCTLR_EL1.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch64 state, MSR or MRS access trapped to EL1.
- SCTLR_EL1.UCI, for execution of cache maintenance instructions using AArch64 state, execution is trapped to EL1 or EL2.
- SCTLR_EL1.UCT, for accesses to CTR_EL0 using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- SCTLR_EL1.DZE, for execution of DC ZVA instructions using AArch64 state, execution is trapped to EL1 or EL2.
- SCTLR_EL1.UMA, for accesses to the PSTATE interrupt masks using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- CPACR_EL1.TTA, for accesses to the trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- MDSCR_EL1.TDCC, for accesses to the Debug Communications Channel (DCC) registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- If FEAT_FGT is implemented, MDCR_EL2.TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and MDCR_EL3.TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- CNTKCTL_EL1.{EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN} accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- PMUSERENR_EL0, for accesses to the Performance Monitor registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- AMUSERENR_EL0.EN, for accesses to Activity Monitors registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- HCR_EL2.{TRVM, TVM}, for accesses to virtual memory control registers using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.TDZ, for execution of DC ZVA instructions using AArch64 state, execution is trapped to EL2.
- HCR_EL2.TTLB, for execution of TLB maintenance instructions using AArch64 state, execution is trapped to EL2.
- HCR_EL2.{TSW, TPC, TPU}, for execution of cache maintenance instructions using AArch64 state, execution is trapped to EL2.
- HCR_EL2.TACR, for accesses to the Auxiliary Control Register, ACTLR_EL1, using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.TIDCP, for accesses to lockdown, DMA, and TCM operations using AArch64 state, MSR or MRS access trapped to EL2.
- If FEAT_TIDCP1 is implemented, SCTLR_EL2.TIDCP, for EL0 accesses to IMPLEMENTATION DEFINED functionality using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.{TID1, TID2, TID3}, for accesses to ID group 1, ID group 2 or ID group 3 registers, using AArch64 state, MSR or MRS access trapped to EL2.
- If FEAT_MTE2 is implemented, HCR_EL2.TID5, for accesses to GMID_EL1, using AArch64 state, MRS or MSR access at EL1, trapped to EL2.
- If FEAT_IDTE3 is implemented, SCR_EL3.TID3, for accesses to ID group 3 registers using AArch64 state, at EL1 and EL2, trapped to EL3.
- CPTR_EL2.TCPAC, for accesses to CPACR_EL1, using AArch64 state, MSR or MRS access trapped to EL2.
- CPTR_EL2.TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL2.

- MDCR_EL2.TTRF, for accesses to the trace filter control register, TRFCR_EL1, using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.TDRA, for accesses to Debug ROM registers, using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.TDOSA, for accesses to powerdown debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- CNTHCTL_EL2.{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.TDA, for accesses to debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- MDCR_EL2.{TPM, TPMCR}, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL2.
- CPTR_EL2.TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL2.
- HCR_EL2.{NV, NV1}, for Nested virtualization register access, using AArch64 state, MSR or MRS access, trapped to EL2.
- HCR_EL2.AT, for execution of AT S1E* instructions, using AArch64 state, execution is trapped to EL2.
- HCR_EL2.{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access, trapped to EL2.
- SCR_EL3.APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL3.
- SCR_EL3.ST, for accesses to the Counter-timer Physical Secure timer registers, using AArch64 state, MSR or MRS access trapped to EL3.
- SCR_EL3.{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL3.
- CPTR_EL3.TCPAC, for accesses to CPTR_EL2 and CPACR_EL1 using AArch64 state, MSR or MRS access trapped to EL3.
- CPTR_EL3.TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TTRF, for accesses to the trace filter control registers, TRFCR_EL1 and TRFCR_EL2, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TDA, for accesses to debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TDOSA, for accesses to powerdown debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- MDCR_EL3.TPM, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL3.
- If FEAT_SPE is implemented:
  - MDCR_EL3.NSPB for accesses to Statistical Profiling and Profiling Buffer control registers, using AArch64 state, MSR or MRS access at EL1 and EL2 trapped to EL3.
  - MDCR_EL2.TPMS for accesses to SPE registers, using AArch64 state, MSR or MRS access at EL1 trapped to EL2.
- CPTR_EL3.TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access, trapped to EL3.
- If FEAT_EVT is implemented, the following registers control traps for EL1 and EL0 Cache controls that use this EC value:
  - HCR_EL2.{TTLBOS, TTLBIS, TICAB, TOCU, TID4}.
  - HCR2.{TTLBIS, TICAB, TOCU, TID4}.
- If FEAT_FGT is implemented:
  - SCR_EL3.FGTEn, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
  - HFGRTR_EL2 for reads and HFGWTR_EL2 for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 trapped to EL2.
  - HFGITR_EL2 for execution of system instructions, MSR or MRS access trapped to EL2.
  - HDFGRTR_EL2 for reads and HDFGWTR_EL2 for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 state trapped to EL2.
  - HAFGRTR_EL2 for reads of Activity Monitor counters, using AArch64 state, MRS access at EL0 and EL1 trapped to EL2.
- If FEAT_RNG_TRAP is implemented, SCR_EL3.TRNDR for reads of RNDR and RNDRRS using AArch64 state, MRS access trapped to EL3.
- If FEAT_SME is implemented:
  - CPTR_EL3.ESM, for MSR or MRS accesses to SMPRI_EL1 at EL1, EL2, and EL3, trapped to EL3.
  - CPTR_EL3.ESM, for MSR or MRS accesses to SMPRIMAP_EL2 at EL2 and EL3, trapped to EL3.
  - SCTLR_EL1.EnTP2, for MSR or MRS accesses to TPIDR2_EL0 at EL0, trapped to EL1 or EL2.
  - SCTLR_EL2.EnTP2, for MSR or MRS accesses to TPIDR2_EL0 at EL0, trapped to EL2.
  - SCR_EL3.EnTP2, for MSR or MRS accesses to TPIDR2_EL0 at EL0, EL1, and EL2, trapped to EL3.
- If FEAT_FPMR is implemented:
  - SCTLR_EL1.EnFPM, for accesses to FPMR at EL0, trapped to EL1 or EL2.
  - SCTLR_EL2.EnFPM, for accesses to FPMR at EL0, trapped to EL2.
  - HCRX_EL2.EnFPM, for accesses to FPMR at EL0 and EL1, trapped to EL2.
  - SCR_EL3.EnFPM, for accesses to FPMR at EL0, EL1, and EL2, trapped to EL3.
- If FEAT_NMI is implemented, HCRX_EL2.TALLINT, for MSR writes of ALLINT at EL1, trapped to EL2.
- If FEAT_FGT2 is implemented:
  - SCR_EL3.FGTEn2, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
  - HFGRTR2_EL2 for reads and HFGWTR2_EL2 for writes of registers, using AArch64 state, using MSR or MRS access at EL1 trapped to EL2.
  - HDFGRTR2_EL2 for reads and HDFGWTR2_EL2 for writes of registers, using AArch64 state, using MSR or MRS access at EL0 and EL1 trapped to EL2.
  - HFGITR2_EL2 for execution of system instructions, MSR or MRS access trapped to EL2.
- If FEAT_ITE is implemented, MDCR_EL3.EnITE, for accesses to Instrumentation trace registers, using AArch64 state, MSR or MRS access, trapped to EL3.
- If FEAT_MEC is implemented, SCR_EL3.MECEn, for accesses to MECID registers at EL2, trapped to EL3.
- If FEAT_SPE_FDS is implemented, MDCR_EL3.EnPMS3 for accesses to SPE registers, using AArch64 state, MSR or MRS access at EL1 and EL2 trapped to EL3.
- If FEAT_SPE_nVM is implemented, MDCR_EL3.EnPMS4 for accesses to SPE registers, using AArch64 state, MSR or MRS access at EL1 and EL2 trapped to EL3.

- If FEAT_RASv2 is implemented, SCR_EL3.TWERR, for accesses to Error Record registers, MSR access at EL1 and EL2 trapped to EL3.
- If FEAT_Debugv8p9 is implemented, MDCR_EL3.EBWE for accesses of MDSELR_EL1, using AArch64 state, MRS or MSR access at EL2 and EL1 trapped to EL3.
- If FEAT_PMUv3p9, FEAT_SPMU, FEAT_EBEP, or FEAT_PMUv3_SS is implemented, MDCR_EL3.EnPM2, for accesses to PMU registers, using AArch64 state, MSR or MRS access at EL2, EL1, and EL0, trapped to EL3.
- If FEAT_PMUv3_SS is implemented, MDCR_EL3.EnPMSS, for accesses to PMU Snapshot registers, using AArch64 state, MSR or MRS access at EL2 and EL1 trapped to EL3.
- If FEAT_THE is implemented, SCR_EL3.RCWMASKEn for accesses to RCWMASK_EL1 and RCWSMASK_EL1, using AArch64 state, MSR or MRS access at EL2 and EL1 trapped to EL3.
- If FEAT_AIE is implemented, SCR_EL3.AIEn for accesses to Extended Memory Attribute registers, MSR or MRS access at EL2 and EL1 trapped to EL3.
- If FEAT_S1PIE, FEAT_S2PIE, FEAT_S1POE, or FEAT_S2POE is implemented, SCR_EL3.PIEn for accesses to Permission Indirection, Overlay registers, MSR or MRS access at EL2, EL1 and EL0 trapped to EL3.
- If FEAT_MPAM_PE_BW_CTRL is implemented, MPAMBW2_EL2.{nTRAP_MPAMBWIDR_EL1, nTRAP_MPAMBW0_EL1, nTRAP_MPAMBW1_EL1} for accesses to MPAMBW*_EL1 registers, using AArch64 state, MRS or MSR access at EL1, trapped to EL2.
- If FEAT_MPAM_PE_BW_CTRL and FEAT_SME are implemented, MPAMBW2_EL2.nTRAP_MPAMBWSM_EL1, for accesses to MPAMBWSM_EL1, using AArch64 state, MRS or MSR access at EL1, trapped to EL2.
- If FEAT_MPAM_PE_BW_CTRL is implemented, MPAMBW3_EL3.nTRAPLOWER, for accesses to MPAMBW_EL2 registers and MPAMBW_EL1 registers, using AArch64 state, MRS or MSR access at EL1, trapped to EL3.
- If FEAT_HACDBS is implemented, SCR_EL3.HACDBSEn, for accesses to HACDBSBR_EL2 and HACDBSCONS_EL2, using AArch64 state, MRS or MSR access at EL2, trapped to EL3.
- If FEAT_HDBSS is implemented, SCR_EL3.HDBSSEn, for accesses to HDBSSBR_EL2 and HDBSSPROD_EL2, using AArch64 state, MRS or MSR access at EL2, trapped to EL3.
- If FEAT_SRMASK is implemented, SCR_EL3.SRMASKEn, for MSR or MRS accesses at EL1 or EL2 to the MASK registers using AArch64 state, trapped to EL3.

ISS encoding for an exception from MSRR, MRRS, or 128-bit System instruction execution in AArch64 state



Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:6]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

Note

This value represents register pair of X[Rt:0], X[Rt:1].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write access, MSRR instructions.
0b1	Read access, MRRS instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from MSRR, MRRS, or 128-bit System instruction execution in AArch64 state

The following fields describe configuration settings for generating exceptions from an MSRR or MRRS access that are reported using EC value 0b010100:

- If FEAT_FGT is implemented:
  - HFGRTR_EL2 for reads and HFGWTR_EL2 for writes of registers, using AArch64 state, accesses at EL1 trapped to EL2.
- If FEAT_FGT2 is implemented:
  - HFGRTR2_EL2.nRCWSMASK_EL1 for reads and HFGWTR2_EL2.nRCWSMASK_EL1 for writes of RCWSMASK_EL1, using AArch64 state, accesses at EL1 trapped to EL2.
- If FEAT_SYSREG128 is implemented:
  - SCTLR2_EL1.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL0 trapped to EL1.
  - SCTLR2_EL2.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL0 trapped to EL2.
  - HCRX_EL2.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL1 and EL0 trapped to EL2.

    - SCR_EL3.EnIDCP128 for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL2, EL1, and EL0 trapped to EL3.
- If FEAT_D128 is implemented:
    - HCR_EL2.{TRVM, TVM} for accesses to TTBR0_EL1 and TTBR1_EL1, accesses at EL1 and EL0 trapped to EL2.
    - HCRX_EL2.D128En for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL1 trapped to EL2.
    - SCR_EL3.D128En for accesses to 128-bit IMPLEMENTATION DEFINED System registers, accesses at EL2 and EL1 trapped to EL3.
- If FEAT_THE is implemented, SCR_EL3.RCWMASKEn for accesses to RCWMASK_EL1 and RCWSMASK_EL1, using AArch64 state, accesses at EL2 and EL1 trapped to EL3.

ISS encoding for an IMPLEMENTATION DEFINED exception to EL3

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMPLEMENTATION DEFINED																								

IMPLEMENTATION DEFINED, bits [24:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from an Instruction Abort

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0										PFV	RES0		SET		FnV	EA	RES0		S1PTW	RES0			IFSC						

When FEAT_S1POE is implemented, if a memory access generates a Instruction Abort due to a Permission fault, the ISS2 encoding for an exception from an Instruction Abort includes further information about the exception.

Bits [24:15]

Reserved, RES0.

PFV, bit [14]

When FEAT_PFAR is implemented and (IFSC == 0b010000, or IFSC IN {0b01001x}, or IFSC IN {0b0101xx}):

PFAR Valid. Describes whether the MFAR_EL3 register is valid.

PFV	Meaning
0b0	MFAR_EL3 is UNKNOWN.
0b1	MFAR_EL3 is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [13]

Reserved, RES0.

SET, bits [12:11]

When FEAT_RAS is implemented and (IFSC == 0b010000, or IFSC IN {0b01001x}, or IFSC IN {0b0101xx}):

Synchronous Error Type. Describes the PE error state after taking the Instruction Abort exception.

SET	Meaning	Applies when
0b00	Recoverable state (UER).	
0b10	Uncontainable (UC).	When FEAT_RASv2 is not implemented
0b11	Restartable state (UEO).	

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External abort exception might result in a PE state that is not recoverable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	FAR is valid.
0b1	FAR is not valid, and holds an UNKNOWN value.

This field is valid only if the IFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

IFSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	When FEAT_RAS is not implemented
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented

0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented

All other values are reserved.

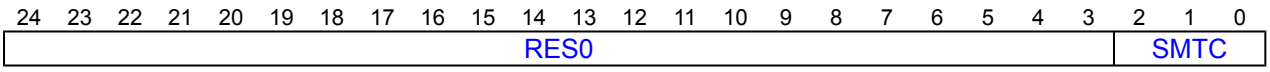
For more information about the lookup level associated with a fault, see 'The lookup level associated with MMU faults'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception due to SME functionality



The accesses covered by this trap include:

- Execution of SME instructions.
- Execution of SVE and Advanced SIMD instructions, when the PE is in Streaming SVE mode.
- Direct accesses of SVCR, SMCR_EL1, SMCR_EL2, SMCR_EL3.

Bits [24:3]

Reserved, RES0.

SMTC, bits [2:0]

SME Trap Code. Identifies the reason for instruction trapping.

SMTC	Meaning	Applies when
0b000	Access to SME functionality trapped as a result of CPACR_EL1.SMEN, CPTR_EL2.SMEN, CPTR_EL2.TSM, or CPTR_EL3.ESM, that is not reported using EC value 0b000000.	
0b001	Advanced SIMD, SVE, or SVE2 instruction trapped because PSTATE.SM is 1.	
0b010	SME instruction trapped because PSTATE.SM is 0.	
0b011	SME instruction trapped because PSTATE.ZA is 0.	
0b100	Access to the SME2 ZT0 register trapped as a result of SMCR_EL1.EZT0, SMCR_EL2.EZT0, or SMCR_EL3.EZT0.	When FEAT_SME2 is implemented

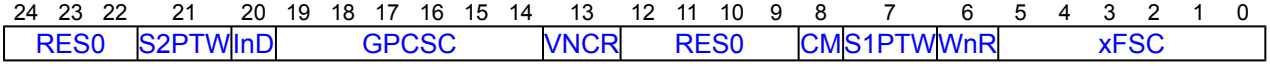
All other values are reserved.

Additional information for the ISS encoding for an exception due to SME functionality

The following fields describe the configuration settings for the traps that are reported using the EC value 0b011101:

- CPACR_EL1.SMEN, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access SVCR and SMCR_EL1 System registers at EL1 and EL0, trapped to EL1 or EL2.
- CPTR_EL2.SMEN and CPTR_EL2.TSM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access SVCR, SMCR_EL1, SMCR_EL2 at EL2, EL1, and EL0, trapped to EL2.
- CPTR_EL3.ESM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access SVCR, SMCR_EL1, SMCR_EL2, SMCR_EL3 from all Exception levels and any Security state, trapped to EL3.
- If FEAT_SME2 is implemented:
  - SMCR_EL1.EZT0, for accesses to ZT0 at EL1 and EL0, trapped to EL1 or EL2.
  - SMCR_EL2.EZT0, for accesses to ZT0 at EL2, EL1, and EL0, trapped to EL2.
  - SMCR_EL3.EZT0, for accesses to ZT0 at any Exception level, trapped to EL3.

ISS encoding for a Granule Protection Check exception



Bits [24:22]

Reserved, RES0.

S2PTW, bit [21]

Indicates whether the Granule Protection Check exception was on an access made for a stage 2 translation table walk.

S2PTW	Meaning
0b0	Fault not on a stage 2 translation table walk.
0b1	Fault on a stage 2 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

InD, bit [20]

Indicates whether the Granule Protection Check exception was on an instruction or data access.

InD	Meaning
0b0	Data access.
0b1	Instruction access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GPCSC, bits [19:14]

Granule Protection Check Status Code.

GPCSC	Meaning
0b000000	GPT address size fault at level 0.
0b000100	GPT walk fault at level 0.
0b000101	GPT walk fault at level 1.
0b001100	Granule protection fault at level 0.
0b001101	Granule protection fault at level 1.
0b010100	Synchronous External abort on GPT fetch at level 0.
0b010101	Synchronous External abort on GPT fetch at level 1.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VNCR, bit [13]

Indicates that the fault came from use of VNCR_EL2 register by EL1 code.

VNCR	Meaning	Applies when
0b0	The fault was not generated by the use of VNCR_EL2 by EL1 code.	
0b1	The fault was generated by the use of VNCR_EL2 by EL1 code.	When FEAT_NV2 is implemented

When InD is 1, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [12:9]

Reserved, RES0.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

CM	Meaning
0b0	The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1.
0b1	The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA, DC GVA, and DC GZVA instructions are not classified as cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

Indicates whether the Granule Protection Check exception was on an access for stage 2 translation for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

WnR	Meaning
0b0	Abort caused by an instruction reading from a memory location.
0b1	Abort caused by an instruction writing to a memory location.

When InD is 1, this field is RES0.

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

xFSC, bits [5:0]

Instruction or Data Fault Status Code.

xFSC	Meaning	Applies when
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented

All other values are reserved.

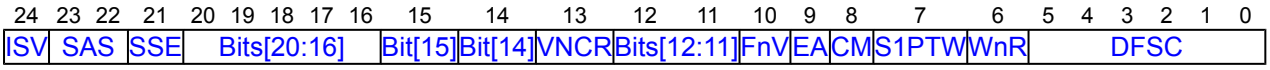
For more information about the lookup level associated with a fault, see 'The lookup level associated with MMU faults'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a Data Abort



The ISS2 encoding for an exception from a Data Abort includes further information about the exception when any of the following features are implemented:

- FEAT_LS64_V.
- FEAT_LS64_ACCDATA.
- FEAT_S1POE.
- FEAT_S1PIE.
- FEAT_GCS.
- FEAT_MTE_CANONICAL_TAGS.

ISV, bit [24]

Instruction Syndrome Valid. Indicates whether the syndrome information in ISS[23:14] is valid.

ISV	Meaning
0b0	No valid instruction syndrome. ISS[23:14] are RES0.
0b1	ISS[23:14] hold a valid instruction syndrome.

In ESR_EL3, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

In ESR_EL3, ISV is 1 when FEAT_LS64_V is implemented and a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

In ESR_EL3, ISV is 1 when FEAT_LS64_ACCDATA is implemented and a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For other faults reported in ESR_EL3, ISV is 0 except for the following stage 2 aborts:

- AArch64 loads and stores of a single general-purpose register (including the register specified with 0b11111, including those with Acquire/Release semantics, but excluding Load Exclusive or Store Exclusive and excluding those with writeback).
- AArch32 instructions where the instruction:
  - Is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
  - Is not performing register writeback.
  - Is not using R15 as a source or destination register.

For these stage 2 aborts, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, and otherwise indicates whether ISS[23:14] hold a valid syndrome.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64_V is implemented and a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64_ACCDATA is implemented and a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

When FEAT_RAS is implemented, ISV is 0 for any synchronous External abort.

When FEAT_RAS is not implemented, it is IMPLEMENTATION DEFINED whether ISV is set to 1 or 0 on a synchronous External abort on a stage 2 translation table walk.

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When FEAT_MTE2 is implemented, for a synchronous Tag Check Fault abort taken to EL3, ESR_EL3.FnV is 0 and FAR_EL3 is valid.

When FEAT_MOPS is implemented, for a synchronous Data Abort on a Memory Copy and Memory Set instruction, ISV is 0.

When FEAT_MTE is implemented, for a synchronous Data Abort on an instruction that directly accesses Allocation Tags, ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SAS, bits [23:22]
When ISV == 1:

Syndrome Access Size. Indicates the size of the access attempted by the faulting operation.

SAS	Meaning
0b00	Byte
0b01	Halfword
0b10	Word
0b11	Doubleword

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0b11.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0b11.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0b11.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSE, bit [21]
When ISV == 1:

Syndrome Sign Extend. For a byte, halfword, or word load operation, indicates whether the data item must be sign extended.

SSE	Meaning
0b0	Sign-extension not required.
0b1	Data item must be sign-extended.

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

For all other operations, this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits[20:16]
When ISV == 1:

SRT, bits [4:0] of bits [20:16]

Syndrome Register Transfer. The register number of the Wt/Xt/Rt operand of the faulting instruction.

If the exception was taken from an Exception level that is using AArch32, then this is the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states'.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ISV == 0, FEAT_RASv2 is implemented, and (DFSC == 0b010000, or DFSC IN {0b01001x}, or DFSC IN {0b0101xx}):

Bits [4:2] of bits [20:16]

Reserved, RES0.

WU, bits [1:0] of bits [20:16]

Write Update. Describes whether a store instruction that generated an External abort updated the location.

WU	Meaning
0b00	Not a store instruction or translation table update, or the location might have been updated.
0b10	Store instruction or translation table update that did not update the location.
0b11	Store instruction or translation table update that updated the location.

In the description of this field, a store instruction is any memory-writing instruction that explicitly performs a store. This includes instructions that both read and write memory.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[15]
When ISV == 1:

SF, bit [15]

Sixty Four bit general-purpose register transfer. Width of the register accessed by the instruction is 64-bit.

SF	Meaning
0b0	Instruction loads/stores a 32-bit general-purpose register.
0b1	Instruction loads/stores a 64-bit general-purpose register.

Note

This field specifies the register width identified by the instruction, not the Execution state.

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ISV == 0:

FnP, bit [15]

FAR not Precise.

FnP	Meaning	Applies when
0b0	The FAR holds the faulting virtual address that generated the Data Abort.	
0b1	The FAR holds any virtual address within the naturally-aligned granule that contains the faulting virtual address that generated a Data Abort due to an SVE contiguous vector load/store instruction, or an SME load/store instruction. For more information about the naturally-aligned fault granule, see FAR_ELx (for example, FAR_EL1).	When FEAT_SME is implemented or FEAT_SVE is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[14]
When ISV == 1:

AR, bit [14]

Acquire/Release.

AR	Meaning
0b0	Instruction did not have acquire/release semantics.
0b1	Instruction did have acquire/release semantics.

When FEAT_LS64 is implemented, if a memory access generated by an LD64B or ST64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_V is implemented, if a memory access generated by an ST64BV instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

When FEAT_LS64_ACCDATA is implemented, if a memory access generated by an ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_PFAR is implemented and (DFSC == 0b010000, or DFSC IN {0b01001x}, or DFSC IN {0b0101xx}):

PFV, bit [14]

PFAR Valid. Describes whether the MFAR_EL3 register is valid.

PFV	Meaning
0b0	MFAR_EL3 is UNKNOWN.
0b1	MFAR_EL3 is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VNCR, bit [13]

Indicates that the fault came from use of VNCR_EL2 register by EL1 code.

VNCR	Meaning	Applies when
0b0	The fault was not generated by the use of VNCR_EL2 by EL1 code.	
0b1	The fault was generated by the use of VNCR_EL2 by EL1 code.	When FEAT_NV2 is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits[12:11]

When (DFSC IN {0b00xxxx} || DFSC IN {0b10101x}) && !(DFSC IN {0b0000xx}):

LST, bits [1:0] of bits [12:11]

Load/Store Type. Used when a Translation fault, Access flag fault, or Permission fault generates a Data Abort.

LST	Meaning	Applies when
0b00	The instruction that generated the Data Abort is not specified by this field.	
0b01	An ST64BV instruction generated the Data Abort.	When FEAT_LS64_V is implemented
0b10	An LD64B or ST64B instruction generated the Data Abort.	When FEAT_LS64 is implemented
0b11	An ST64BV0 instruction generated the Data Abort.	When FEAT_LS64_ACCDATA is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_RAS is implemented and (DFSC == 0b010000, or DFSC IN {0b01001x}, or DFSC IN {0b0101xx}):

SET, bits [1:0] of bits [12:11]

Synchronous Error Type. Used when a synchronous External abort, not on a Translation table walk or hardware update of the Translation table, generated the Data Abort. Describes the PE error state after taking the Data Abort exception.

SET	Meaning	Applies when
0b00	Recoverable state (UER).	
0b10	Uncontainable (UC).	When FEAT_RASv2 is not implemented
0b11	Restartable state (UEO).	

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External abort exception might result in a PE state that is not recoverable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	FAR is valid.
0b1	FAR is not valid, and holds an UNKNOWN value.

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

CM	Meaning
0b0	The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1.
0b1	The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA, DC GVA, and DC GZVA instructions are not classified as cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

WnR	Meaning
0b0	Abort caused by an instruction reading from a memory location.
0b1	Abort caused by an instruction writing to a memory location.

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- If FEAT_RASv2 is implemented, an External abort on an Atomic access, reported with ESR_EL3.WU set to `0b00`.
- A fault reported using a DFSC value of `0b110101` or `0b110001`, indicating an unsupported Exclusive or atomic access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

DFSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Synchronous Tag Check Fault.	When FEAT_MTE2 is implemented
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	When FEAT_RAS is not implemented
0b100001	Alignment fault.	

0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented
0b110100	IMPLEMENTATION DEFINED fault (Lockdown).	
0b110101	IMPLEMENTATION DEFINED fault (Unsupported Exclusive or Atomic access).	

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The lookup level associated with MMU faults'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a trapped floating-point exception

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0	TFV	RES0												VECITR			IDF	RES0		IXF	UFF	OFF	DZF	IOF

Bit [24]

Reserved, RES0.

TFV, bit [23]

Trapped Fault Valid bit. Indicates whether the IDF, IXF, UFF, OFF, DZF, and IOF bits hold valid information about trapped floating-point exceptions.

TFV	Meaning
0b0	The IDF, IXF, UFF, OFF, DZF, and IOF bits do not hold valid information about trapped floating-point exceptions and are UNKNOWN.
0b1	One or more floating-point exceptions occurred during an operation performed while executing the reported instruction. The IDF, IXF, UFF, OFF, DZF, and IOF bits indicate trapped floating-point exceptions that occurred. For more information, see 'Floating-point exceptions and exception traps'.

It is IMPLEMENTATION DEFINED whether this field is set to 0 on an exception generated by a trapped floating-point exception from an instruction that is performing floating-point operations on more than one lane of a vector.

Note

This is not a requirement. Implementations can set this field to 1 on a trapped floating-point exception from an instruction and return valid information in the {IDF, IXF, UFF, OFF, DZF, IOF} fields.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [22:11]

Reserved, RES0.

VECITR, bits [10:8]

For a trapped floating-point exception from an instruction executed in AArch32 state this field is RES1.

For a trapped floating-point exception from an instruction executed in AArch64 state this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IDF	Meaning
0b0	Input denormal floating-point exception has not occurred.
0b1	Input denormal floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IXF	Meaning
0b0	Inexact floating-point exception has not occurred.
0b1	Inexact floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

UFF	Meaning
0b0	Underflow floating-point exception has not occurred.
0b1	Underflow floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

OFF	Meaning
0b0	Overflow floating-point exception has not occurred.
0b1	Overflow floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

DZF	Meaning
0b0	Divide by Zero floating-point exception has not occurred.
0b1	Divide by Zero floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

IOF	Meaning
0b0	Invalid Operation floating-point exception has not occurred.
0b1	Invalid Operation floating-point exception occurred during execution of the reported instruction.

The reset behavior of this field is:

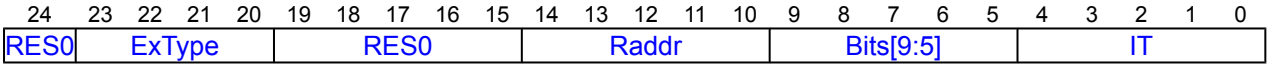
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from a trapped floating-point exception

In an implementation that supports the trapping of floating-point exceptions:

- From an Exception level using AArch64, the FPCR.{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.
- From an Exception level using AArch32, the FPSCR.{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.

ISS encoding for a GCS exception



Bit [24]

Reserved, RES0.

ExType, bits [23:20]

The first level classification of GCS exceptions.

ExType	Meaning
0b0000	The exception reported is a Guarded Control Stack Data Check Exception.
0b0001	The exception reported is an EXLOCK Exception.
0b0010	The exception reported is a trap exception on GCSSTR or GCSSTTR instruction execution.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:15]

Reserved, RES0.

Raddr, bits [14:10]

When ExType == 0b0010 :

Indicates the data address register number supplied in the instruction that has been trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits[9:5]

When ExType == 0b0000 :

Rn, bits [4:0] of bits [9:5]

Indicates a register number used by the instruction that caused the Guarded Control Stack Data Check Exception.

For a procedure return instruction reported with ESR_EL3.ISS.IT as 0b00000, contains the register number for the register which contains the target address of the branch.

For a GCSPOPM instruction reported with ESR_EL3.ISS.IT as 0b00001, contains the register number for the register which is the destination register of the instruction.

For a procedure return instruction reported with ESR_EL3.ISS.IT as `0b00010` or `0b00011`, contains the value `0b11110`, indicating X30.

For a GCSSS1 instruction reported with ESR_EL3.ISS.IT as `0b00100`, contains the register number for the register which is the input register of the instruction.

If ESR_EL3.ISS.IT is reported as `0b00101` or `0b01000`, this field is UNKNOWN

If ESR_EL3.ISS.IT is reported as `0b01001`, this field is `0b11111`

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ExType == 0b0010 :

Rvalue, bits [4:0] of bits [9:5]

Indicates the data value register number supplied in the instruction that has been trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IT, bits [4:0]

When ExType == 0b0000 :

Type of the instruction that caused the Guarded Control Stack Data Check Exception.

IT	Meaning
`0b00000`	Guarded Control Stack Data Check Exception is from a procedure return instruction without Pointer authentication.
`0b00001`	Guarded Control Stack Data Check Exception is from a GCSPOPM instruction.
`0b00010`	Guarded Control Stack Data Check Exception is from a procedure return instruction with Pointer authentication that uses key A.
`0b00011`	Guarded Control Stack Data Check Exception is from a procedure return instruction with Pointer authentication that uses key B.
`0b00100`	Guarded Control Stack Data Check Exception is from a GCSSS1 instruction.
`0b00101`	Guarded Control Stack Data Check Exception is from a GCSSS2 instruction.
`0b01000`	Guarded Control Stack Data Check Exception is from a GCSPOPCX instruction.
`0b01001`	Guarded Control Stack Data Check Exception is from a GCSPOPX instruction.

All other values are reserved

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

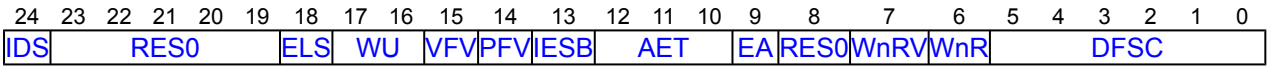
Reserved, RES0.

Additional information for the ISS encoding for a GCS exception

The following fields describe the configuration settings for the traps that are reported using EC value `0b101101` and ExType value `0b0010`:

- GCSCRE0_EL1.STREn
- GCSCR_EL1.STREn.
- GCSCR_EL2.STREn.
- GCSCR_EL3.STREn.
- HFGITR_EL2.nGCSSTR_EL1.

ISS encoding for an SError exception



Note

In earlier versions of the architecture, an SError exception is referred to as an SError interrupt or an asynchronous External abort exception.

IDS, bit [24]

IMPLEMENTATION DEFINED syndrome.

IDS	Meaning
`0b0`	Bits [23:0] of the ISS field holds the fields described in this encoding. **Note** If FEAT_RAS is not implemented, bits [23:0] of the ISS field are RES0.
`0b1`	Bits [23:0] of the ISS field holds IMPLEMENTATION DEFINED syndrome information that can be used to provide additional information about the SError exception.

Note

This field was previously called ISV.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:19]

Reserved, RES0.

ELS, bit [18]

When FEAT_RASv2 is implemented and DFSC == 0b010001:

Meaning of ELR_ELx.

ELS	Meaning
`0b0`	Asynchronous. Does not indicate the trigger for the exception.
`0b1`	Synchronous. The exception was triggered by the instruction at ELR_ELx.

SError exceptions that report this field is 1 are not required to be precise.

The ESR_EL3.AET field describes whether the exception is precise or imprecise.

Corrected, Recoverable or Restartable exceptions are precise. Unrecoverable or Uncontainable exceptions are imprecise.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WU, bits [17:16]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

Write Update. Describes whether a store instruction that generated an External abort updated the location.

WU	Meaning
0b00	Not a store instruction or translation table update, or the location might have been updated.
0b10	Store instruction or translation table update that did not update the location.
0b11	Store instruction or translation table update that updated the location.

In the description of this field, a store instruction is any memory-writing instruction that explicitly performs a store. This includes instructions that both read and write memory.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VFV, bit [15]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

FAR Valid. Indicates the FAR_EL3 register contains a valid virtual address.

VFV	Meaning
0b0	FAR_EL3 is not valid, and holds an UNKNOWN value.
0b1	FAR_EL3 contains a valid virtual address associated with the error.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PFV, bit [14]
When FEAT_PFAR is implemented and DFSC == 0b010001:

PFAR Valid. Describes whether the MFAR_EL3 register is valid.

PFV	Meaning
0b0	MFAR_EL3 is UNKNOWN.
0b1	MFAR_EL3 is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IESB, bit [13]
When FEAT_IESB is implemented and DFSC == 0b010001:

Implicit error synchronization event.

IESB	Meaning
0b0	The SError exception was either not synchronized by the implicit error synchronization event or not taken immediately.
0b1	The SError exception was synchronized by the implicit error synchronization event and taken immediately.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AET, bits [12:10]
When FEAT_RAS is implemented and DFSC == 0b010001:

Asynchronous Error Type.

Describes the PE error state after taking the SError exception.

AET	Meaning
0b000	Uncontainable (UC).
0b001	Unrecoverable state (UEU).
0b010	Restartable state (UEO).
0b011	Recoverable state (UER).
0b110	Corrected (CE).

All other values are reserved.

If multiple errors are taken as a single SError exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA, bit [9]
When FEAT_RAS is implemented and DFSC == 0b010001:

External abort type. Provides an IMPLEMENTATION DEFINED classification of External aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

WnRV, bit [7]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

ESR_EL3.WnR valid.

WnRV	Meaning
0b0	ESR_EL3.WnR is not valid and has been set to 0b0.
0b1	ESR_EL3.WnR is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WnR, bit [6]
When FEAT_RASv2 is implemented and DFSC == 0b010001:

Write-not-Read. When the WnRV field is 0b1, indicates whether an exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

WnR	Meaning
0b0	Exception was caused by an instruction reading from a memory location.
0b1	Exception was caused by an instruction writing to a memory location.

Accessing this bit has the following behavior:

- This bit is RES0 if ESR_EL3.WnRV==0b0.
- This bit is not valid and reads UNKNOWN if an External abort on a Atomic access, reported with ESR_EL3.WU == 0b00.
- Otherwise RW.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DFSC, bits [5:0]

When FEAT_RAS is implemented:

Data Fault Status Code.

DFSC	Meaning
0b000000	Uncategorized error.
0b010001	Asynchronous SError exception.

All other values are reserved.

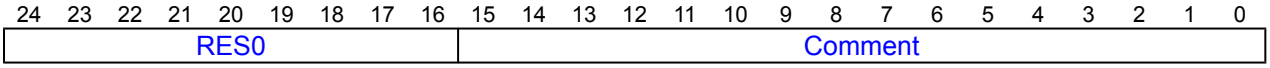
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ISS encoding for an exception from execution of a Breakpoint instruction



Bits [24:16]

Reserved, RES0.

Comment, bits [15:0]

Set to the instruction comment field value, zero extended as necessary.

For the AArch32 BKPT instructions, the comment field is described as the immediate field.

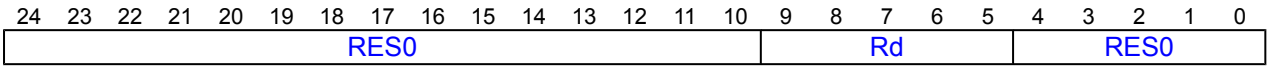
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for an exception from execution of a Breakpoint instruction

For more information about generating these exceptions, see 'Breakpoint instruction exceptions'.

ISS encoding for an exception from a TSTART instruction



Bits [24:10]

Reserved, RES0.

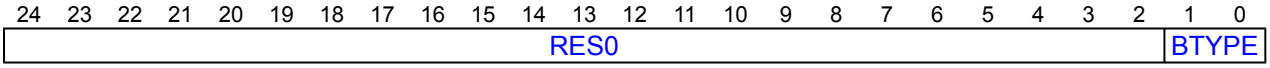
Rd, bits [9:5]

The Rd value from the issued instruction, the general purpose register used for the destination.

Bits [4:0]

Reserved, RES0.

ISS encoding for an exception from Branch Target Identification instruction



Bits [24:2]

Reserved, RES0.

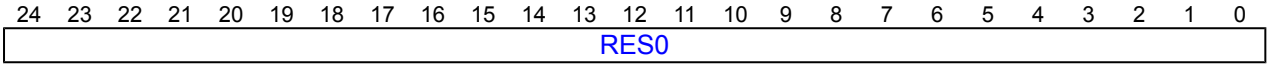
BTYPE, bits [1:0]

This field is set to the PSTATE.BTYPE value that generated the Branch Target Exception.

Additional information for the ISS encoding for an exception from Branch Target Identification instruction

For more information about generating these exceptions, see 'The AArch64 application level programmers model'.

ISS encoding for an exception from a trapped Pointer Authentication instruction



Bits [24:0]

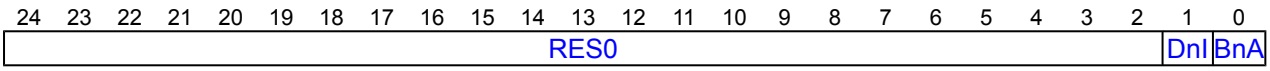
Reserved, RES0.

Additional information for the ISS encoding for an exception from a trapped Pointer Authentication instruction

For more information about generating these exceptions, see:

- HCR_EL2.API, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL2.
- SCR_EL3.API, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL3.

ISS encoding for a PAC Fail exception



Bits [24:2]

Reserved, RES0.

DnI, bit [1]

This field indicates whether the exception is as a result of an Instruction key or a Data key.

DnI	Meaning
0b0	Instruction Key.
0b1	Data Key.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BnA, bit [0]

This field indicates whether the exception is as a result of an A key or a B key.

BnA	Meaning
0b0	A key.
0b1	B key.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for a PAC Fail exception

The following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:

- AUTDA, AUTDZA.
- AUTDB, AUTDZB.
- AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIZA.
- AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZB.
- AUTIASPPC, AUTIASPPCR, AUTIA171615.
- AUTIBSPPC, AUTIBSPPCR, AUTIB171615.

If FEAT_FPACCOMBINE is implemented, the following instructions generate a PAC Fail exception when the Pointer Authentication Code (PAC) is incorrect:

- RETAA, RETAB.
- RETAASPPC, RETABSPPC.
- RETAASPPCR, RETABSPPCR.
- BLRAA, BLRAAZ, BLRAB, BLRABZ.
- BRAA, BRAB, BRAAZ, BRABZ.
- ERETAA, ERETAB.
- LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

Accessing ESR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ESR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0010	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ESR_EL3;
```

```
MSR ESR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0010	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ESR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FAR_EL1, Fault Address Register (EL1)

The FAR_EL1 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction Abort exceptions, Data Abort exceptions, PC alignment fault exceptions and Watchpoint exceptions that are taken to EL1.

Configuration

AArch64 System register FAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DFAR[31:0].

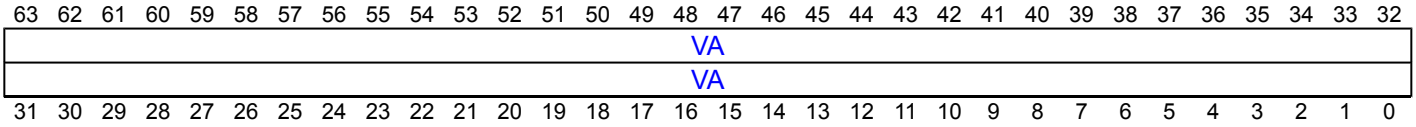
AArch64 System register FAR_EL1 bits [63:32] are architecturally mapped to AArch32 System register IFAR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to FAR_EL1 are UNDEFINED.

Attributes

FAR_EL1 is a 64-bit register.

Field descriptions



VA, bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL1. Exceptions that set the FAR_EL1 are Instruction Aborts (EC `0x20` or `0x21`), Data Aborts (EC `0x24` or `0x25`), PC alignment faults (EC `0x22`), and Watchpoints (EC `0x34` or `0x35`). ESR_EL1.EC holds the EC syndrome value for the exception.

For a synchronous External abort:

- If the VA that generated the abort was from an address range for which Address tagging is enabled for the translation regime in use when the abort was generated, then bits[63:56] of FAR_EL1 are UNKNOWN.
- If the VA that generated the abort was from an address range for which Address tagging is disabled and Logical Address Tagging is enabled for the translation regime in use when the abort was generated, then bits[59:56] of FAR_EL1 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL1.FnV is 0, and FAR_EL1 is UNKNOWN if ESR_EL1.FnV is 1.

On an exception due to a Tag Check Fault caused by a data cache maintenance or other DC instruction, the address held in FAR_EL1 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument of the instruction as generated by MMU faults caused by DC ZVA.

If a memory fault that sets FAR_EL1 is generated from an STZGM instruction, the address held in FAR_EL1 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument.

If a memory fault that sets FAR_EL1, other than a Tag Check Fault, is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

If the exception that updates FAR_EL1 is taken from an Exception level using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are `0x00000001`:

- The faulting address was generated by a load or store instruction that sequentially incremented from address `0xFFFFFFFF`. Such a load or store instruction is CONSTRAINED UNPREDICTABLE.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

When the PE sets ESR_EL1.{ISV,FnP} to {0,1} on taking a Data Abort exception, the PE sets FAR_EL1 to any address within the naturally-aligned fault granule that contains the virtual address of the memory access that generated the Data Abort exception.

When the PE sets ESR_EL1.{FnV,FnP} to {0,1} on taking a Watchpoint exception, the PE sets FAR_EL1 to any address within the naturally-aligned fault granule that contains the virtual address of the memory access that generated the Watchpoint exception.

The naturally-aligned fault granule is one of:

- When ESR_EL1.DFSC is `0b010001`, indicating a Synchronous Tag Check fault, it is a 16-byte tag granule.
- When ESR_EL1.DFSC is `0b11010x`, indicating an IMPLEMENTATION DEFINED fault, it is an IMPLEMENTATION DEFINED granule.
- Otherwise, it is the smallest implemented translation granule.

When FEAT_MOPS is implemented, the value in FAR_EL1 on a synchronous exception from any of the Memory Copy and Memory Set instructions represents the first element that has not been copied or set, and is determined as follows:

- For a Data Abort generated by the MMU, the value is within the address range of the relevant translation granule, aligned to the size of the relevant translation granule of the address that generated the Data Abort. Bits[(n-1):0] of the value are UNKNOWN, where 2^n is the relevant translation granule size in bytes. For the purpose of calculating the relevant translation granule, if the MMU is disabled for a stage of translation, then the current translation granule size is equal to 2^{64} for stage 1, and the PARange for stage 2. The relevant translation granule is:
  - For MMU faults generated at stage 1, the current stage 1 translation granule.
  - For MMU faults generated at stage 2, the smaller of the current stage 1 translation granule and the current stage 2 translation granule.
  - If FEAT_RME is implemented, for a synchronous Data Abort generated as the result of a GPF, the smallest of the current stage 1 translation granule, the current stage 2 translation granule and the configured granule size in GPCCR_EL3.PGS.
- For a Data Abort generated by a Tag Check Fault, the value is any address that caused a Tag Check Fault within the block size of the load or store.
- For a Watchpoint exception, the value is an address range of the size defined by the DCZID_EL0.BS field. This address does not need to be the element with a watchpoint, but can be some earlier element.
- Otherwise, the value is the lowest address in the block size of the load or store.

For a Data Abort exception or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging'.

For a synchronous Tag Check Fault:

- If FEAT_MTE_TAGGED_FAR is implemented or Address tagging is disabled for the translation regime in use when the abort was generated, all bits of FAR_EL1 are not UNKNOWN.
- If FEAT_MTE_TAGGED_FAR is not implemented and Address tagging is enabled for the translation regime in use when the abort was generated, bits[63:60] of FAR_EL1 are UNKNOWN.

Execution at EL0 makes FAR_EL1 become UNKNOWN.

Note

The address held in this field is an address accessed by the instruction fetch or data access that caused the exception that actually gave rise to the instruction or Data Abort. It is the lower address that gave rise to the fault that is reported. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize which fault is reported.

For all other exceptions taken to EL1, FAR_EL1 is UNKNOWN.

FAR_EL1 is made UNKNOWN on an exception return from EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FAR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name FAR_EL1 or FAR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, FAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x220];
    else
        X[t, 64] = FAR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = FAR_EL2;
    else
        X[t, 64] = FAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = FAR_EL1;
```

```
MSR FAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x220] = X[t, 64];
    else
        FAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        FAR_EL2 = X[t, 64];
    else
        FAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    FAR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, FAR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x220];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = FAR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = FAR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR FAR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x220] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        FAR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        FAR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MRS <Xt>, FAR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = FAR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = FAR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = FAR_EL2;
```

When FEAT_VHE is implemented

```
MSR FAR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        FAR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    FAR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    FAR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FAR_EL2, Fault Address Register (EL2)

The FAR_EL2 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction Abort exceptions, Data Abort exceptions, PC alignment fault exceptions and Watchpoint exceptions that are taken to EL2.

Configuration

AArch64 System register FAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HDFAR[31:0].

AArch64 System register FAR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HIFAR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to FAR_EL2 are UNDEFINED.

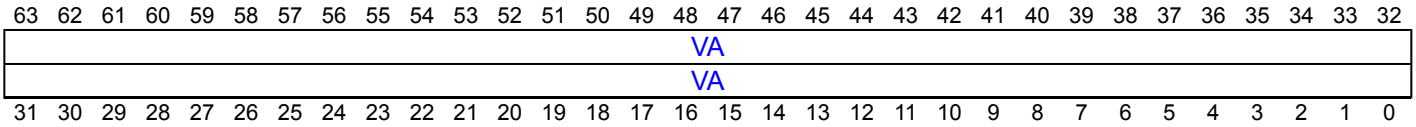
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

FAR_EL2 is a 64-bit register.

Field descriptions



VA, bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL2. Exceptions that set the FAR_EL2 are Instruction Aborts (EC `0x20` or `0x21`), Data Aborts (EC `0x24` or `0x25`), PC alignment faults (EC `0x22`), and Watchpoints (EC `0x34` or `0x35`). ESR_EL2.EC holds the EC syndrome value for the exception.

For a synchronous External abort:

- If the VA that generated the abort was from an address range for which Address tagging is enabled for the translation regime in use when the abort was generated, then bits[63:56] of FAR_EL2 are UNKNOWN.
- If the VA that generated the abort was from an address range for which Address tagging is disabled and Logical Address Tagging is enabled for the translation regime in use when the abort was generated, then bits[59:56] of FAR_EL2 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL2.FnV is 0, and FAR_EL2 is UNKNOWN if ESR_EL2.FnV is 1.

On an exception due to a Tag Check Fault caused by a data cache maintenance or other DC instruction, the address held in FAR_EL2 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument of the instruction as generated by MMU faults caused by DC ZVA.

If a memory fault that sets FAR_EL2 is generated from an STZGM instruction, the address held in FAR_EL2 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument.

If a memory fault that sets FAR_EL2, other than a Tag Check Fault, is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

If the exception that updates FAR_EL2 is taken from an Exception level using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are `0x00000001`:

- The faulting address was generated by a load or store instruction that sequentially incremented from address `0xFFFFFFFF`. Such a load or store instruction is CONSTRAINED UNPREDICTABLE.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

When the PE sets ESR_EL2.{ISV,FnP} to {0,1} on taking a Data Abort exception, the PE sets FAR_EL2 to any address within the naturally-aligned fault granule that contains the virtual address of the memory access that generated the Data Abort exception.

When the PE sets ESR_EL2.{FnV,FnP} to {0,1} on taking a Watchpoint exception, the PE sets FAR_EL2 to any address within the naturally-aligned fault granule that contains the virtual address of the memory access that generated the Watchpoint exception.

The naturally-aligned fault granule is one of:

- When ESR_EL2.DFSC is `0b010001`, indicating a Synchronous Tag Check fault, it is a 16-byte tag granule.
- When ESR_EL2.DFSC is `0b11010x`, indicating an IMPLEMENTATION DEFINED fault, it is an IMPLEMENTATION DEFINED granule.
- Otherwise, it is the smallest implemented translation granule.

When FEAT_MOPS is implemented, the value in FAR_EL2 on a synchronous exception from any of the Memory Copy and Memory Set instructions represents the first element that has not been copied or set, and is determined as follows:

- For a Data Abort generated by the MMU, the value is within the address range of the relevant translation granule, aligned to the size of the relevant translation granule of the address that generated the Data Abort. Bits[(n-1):0] of the value are UNKNOWN, where 2^n is the relevant translation granule size in bytes. For the purpose of calculating the relevant translation granule, if the MMU is disabled for a stage of translation, then the current translation granule size is equal to 2^{64} for stage 1, and the PARange for stage 2. The relevant translation granule is:
  - For MMU faults generated at stage 1, the current stage 1 translation granule.
  - For MMU faults generated at stage 2, the smaller of the current stage 1 translation granule and the current stage 2 translation granule.
  - If FEAT_RME is implemented, for a synchronous Data Abort generated as the result of a GPF, the smallest of the current stage 1 translation granule, the current stage 2 translation granule and the configured granule size in GPCCR_EL3.PGS.
- For a Data Abort generated by a Tag Check Fault, the value is any address that caused a Tag Check Fault within the block size of the load or store.
- For a Watchpoint exception, the value is an address range of the size defined by the DCZID_EL0.BS field. This address does not need to be the element with a watchpoint, but can be some earlier element.
- Otherwise, the value is the lowest address in the block size of the load or store.

For a Data Abort exception or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging'.

For a synchronous Tag Check Fault:

- If FEAT_MTE_TAGGED_FAR is implemented or Address tagging is disabled for the translation regime in use when the abort was generated, all bits of FAR_EL2 are not UNKNOWN.
- If FEAT_MTE_TAGGED_FAR is not implemented and Address tagging is enabled for the translation regime in use when the abort was generated, bits[63:60] of FAR_EL2 are UNKNOWN.

Execution at EL1 or EL0 makes FAR_EL2 become UNKNOWN.

Note

The address held in this field is an address accessed by the instruction fetch or data access that caused the exception that actually gave rise to the instruction or Data Abort. It is the lower address that gave rise to the fault that is reported. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize which fault is reported.

For all other exceptions taken to EL2, FAR_EL2 is UNKNOWN.

FAR_EL2 is made UNKNOWN on an exception return from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FAR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name FAR_EL2 or FAR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, FAR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = FAR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = FAR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = FAR_EL2;
```

```
MSR FAR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        FAR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    FAR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    FAR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, FAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x220];
    else
        X[t, 64] = FAR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = FAR_EL2;
    else
        X[t, 64] = FAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = FAR_EL1;
```

When FEAT_VHE is implemented

```
MSR FAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x220] = X[t, 64];
    else
        FAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        FAR_EL2 = X[t, 64];
    else
        FAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    FAR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FAR_EL3, Fault Address Register (EL3)

The FAR_EL3 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction Abort exceptions, Data Abort exceptions and PC alignment fault exceptions that are taken to EL3.

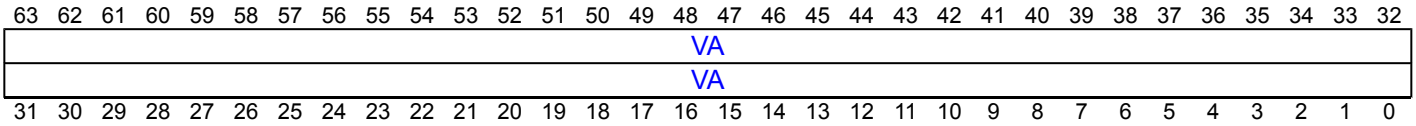
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to FAR_EL3 are UNDEFINED.

Attributes

FAR_EL3 is a 64-bit register.

Field descriptions



VA, bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL3. Exceptions that set the FAR_EL3 are Instruction Aborts (EC `0x20` or `0x21`), Data Aborts (EC `0x24` or `0x25`), and PC alignment faults (EC `0x22`). ESR_EL3.EC holds the EC syndrome value for the exception.

For a synchronous External abort:

- If the VA that generated the abort was from an address range for which Address tagging is enabled for the translation regime in use when the abort was generated, then bits[63:56] of FAR_EL3 are UNKNOWN.
- If the VA that generated the abort was from an address range for which Address tagging is disabled and Logical Address Tagging is enabled for the translation regime in use when the abort was generated, then bits[59:56] of FAR_EL3 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL3.FnV is 0, and FAR_EL3 is UNKNOWN if ESR_EL3.FnV is 1.

On an exception due to a Tag Check Fault caused by a data cache maintenance or other DC instruction, the address held in FAR_EL3 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument of the instruction as generated by MMU faults caused by DC ZVA.

If a memory fault that sets FAR_EL3 is generated from an STZGM instruction, the address held in FAR_EL3 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument.

If a memory fault that sets FAR_EL3, other than a Tag Check Fault, is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

If the exception that updates FAR_EL3 is taken from an Exception level using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are `0x00000001`:

- The faulting address was generated by a load or store instruction that sequentially incremented from address `0xFFFFFFFF`. Such a load or store instruction is CONSTRAINED UNPREDICTABLE.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

When the PE sets ESR_EL3.{ISV,FnP} to {0,1} on taking a Data Abort exception, the PE sets FAR_EL3 to any address within the naturally-aligned fault granule that contains the virtual address of the memory access that generated the Data Abort exception.

The naturally-aligned fault granule is one of:

- When ESR_EL3.DFSC is `0b010001`, indicating a Synchronous Tag Check fault, it is a 16-byte tag granule.
- When ESR_EL3.DFSC is `0b11010x`, indicating an IMPLEMENTATION DEFINED fault, it is an IMPLEMENTATION DEFINED granule.
- Otherwise, it is the smallest implemented translation granule.

When FEAT_MOPS is implemented, the value in FAR_EL3 on a synchronous exception from any of the Memory Copy and Memory Set instructions represents the first element that has not been copied or set, and is determined as follows:

- For a Data Abort generated by the MMU, the value is within the address range of the relevant translation granule, aligned to the size of the relevant translation granule of the address that generated the Data Abort. Bits[(n-1):0] of the value are UNKNOWN, where 2^n is the relevant translation granule size in bytes. For the purpose of calculating the relevant translation granule, if the MMU is disabled for a stage of translation, then the current translation granule size is equal to 2^{64} for stage 1, and the PARange for stage 2. The relevant translation granule is:
  - For MMU faults generated at stage 1, the current stage 1 translation granule.
  - For MMU faults generated at stage 2, the smaller of the current stage 1 translation granule and the current stage 2 translation granule.
  - If FEAT_RME is implemented, for a synchronous Data Abort generated as the result of a GPF, the smallest of the current stage 1 translation granule, the current stage 2 translation granule and the configured granule size in GPCCR_EL3.PGS.
- For a Data Abort generated by a Tag Check Fault, the value is any address that caused a Tag Check Fault within the block size of the load or store.
- Otherwise, the value is the lowest address in the block size of the load or store.

For a Data Abort exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging'.

For a synchronous Tag Check Fault:

- If FEAT_MTE_TAGGED_FAR is implemented or Address tagging is disabled for the translation regime in use when the abort was generated, all bits of FAR_EL3 are not UNKNOWN.
- If FEAT_MTE_TAGGED_FAR is not implemented and Address tagging is enabled for the translation regime in use when the abort was generated, bits[63:60] of FAR_EL3 are UNKNOWN.

Execution at EL2, EL1, and EL0 makes FAR_EL3 become UNKNOWN.

Note

The address held in this register is an address accessed by the instruction fetch or data access that caused the exception that actually gave rise to the instruction or Data Abort. It is the lowest address that gave rise to the fault that is reported. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize which fault is reported.

For all other exceptions taken to EL3, FAR_EL3 is UNKNOWN.

FAR_EL3 is made UNKNOWN on an exception return from EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FAR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, FAR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0110	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = FAR_EL3;
```

```
MSR FAR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0110	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    FAR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FGWTE3_EL3, Fine-Grained Write Traps EL3

The FGWTE3_EL3 characteristics are:

Purpose

Provides controls for traps of MSR and MSRR writes to specified EL3 system registers.

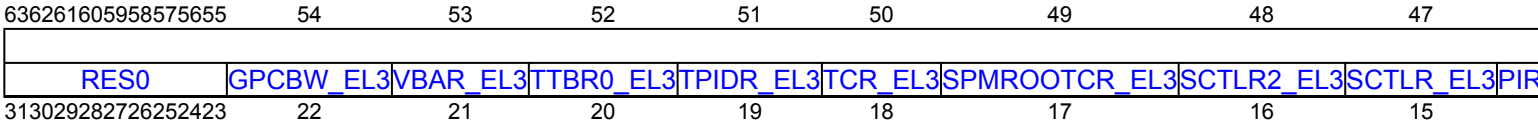
Configuration

This register is present only when EL3 is implemented, FEAT_FGWTE3 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to FGWTE3_EL3 are UNDEFINED.

Attributes

FGWTE3_EL3 is a 64-bit register.

Field descriptions



MSR accesses are trapped to EL3 and reported with EC syndrome value 0x18.

MSRR accesses are trapped to EL3 and reported with EC syndrome value 0x14.

The bits in this register are sticky. Writes to these bits have the following properties:

- A write of 0b0 is ignored.
- A write of 0b1 updates the bit to 0b1.

Bits [63:23]

Reserved, RES0.

GPCBW_EL3, bit [22]

When FEAT_RME_GPC3 is implemented:

Traps accesses of GPCBW_EL3 to EL3.

GPCBW_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

VBAR_EL3, bit [21]

Traps accesses of VBAR_EL3 to EL3.

VBAR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

TTBR0_EL3, bit [20]

Traps accesses of TTBR0_EL3 to EL3.

TTBR0_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

TPIDR_EL3, bit [19]

Traps accesses of TPIDR_EL3 to EL3.

TPIDR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

TCR_EL3, bit [18]

Traps accesses of TCR_EL3 to EL3.

TCR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

SPMROOTCR_EL3, bit [17]

When FEAT_RME is implemented and FEAT_SPMU is implemented:

Traps accesses of SPMROOTCR_EL3 to EL3.

SPMROOTCR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

SCTLR2_EL3, bit [16]

When FEAT_SCTLR2 is implemented:

Traps accesses of SCTLR2_EL3 to EL3.

SCTLR2_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

SCTLR_EL3, bit [15]

Traps accesses of SCTLR_EL3 to EL3.

SCTLR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

PIR_EL3, bit [14]

When FEAT_S1PIE is implemented:

Traps accesses of PIR_EL3 to EL3.

PIR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

MPAM3_EL3, bit [13]

When FEAT_MPAM is implemented:

Traps accesses of MPAM3_EL3 to EL3.

MPAM3_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

MECID_RL_A_EL3, bit [12]

When FEAT_MEC is implemented:

Traps accesses of MECID_RL_A_EL3 to EL3.

MECID_RL_A_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

MDCR_EL3, bit [11]

Traps accesses of MDCR_EL3 to EL3.

MDCR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

MAIR2_EL3, bit [10]

When FEAT_AIE is implemented:

Traps accesses of MAIR2_EL3 to EL3.

MAIR2_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

MAIR_EL3, bit [9]

Traps accesses of MAIR_EL3 to EL3.

MAIR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

GPTBR_EL3, bit [8]

When FEAT_RME is implemented:

Traps accesses of GPTBR_EL3 to EL3.

GPTBR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

GPCCR_EL3, bit [7]

When FEAT_RME is implemented:

Traps accesses of GPCCR_EL3 to EL3.

GPCCR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

GCSPR_EL3, bit [6]

When FEAT_GCS is implemented:

Traps accesses of GCSPR_EL3 to EL3.

GCSPR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

GCSCR_EL3, bit [5]

When FEAT_GCS is implemented:

Traps accesses of GCSCR_EL3 to EL3.

GCSCR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

AMAIR2_EL3, bit [4]

When FEAT_AIE is implemented:

Traps accesses of AMAIR2_EL3 to EL3.

AMAIR2_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Otherwise:

Reserved, RES0.

AMAIR_EL3, bit [3]

Traps accesses of AMAIR_EL3 to EL3.

AMAIR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

AFSR1_EL3, bit [2]

Traps accesses of AFSR1_EL3 to EL3.

AFSR1_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

AFSR0_EL3, bit [1]

Traps accesses of AFSR0_EL3 to EL3.

AFSR0_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

ACTLR_EL3, bit [0]

Traps accesses of ACTLR_EL3 to EL3.

ACTLR_EL3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	MSR write accesses to the specified register are trapped to EL3 with EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1S**.

Accessing FGWTE3_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, FGWTE3_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b101

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_FGWTE3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = FGWTE3_EL3;
```

MSR FGWTE3_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b101

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_FGWTE3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    FGWTE3_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FPCR, Floating-point Control Register

The FPCR characteristics are:

Purpose

Controls floating-point behavior.

Configuration

AArch64 System register FPCR bits [26:15] are architecturally mapped to AArch32 System register FPSCR[26:15].

AArch64 System register FPCR bits [12:8] are architecturally mapped to AArch32 System register FPSCR[12:8].

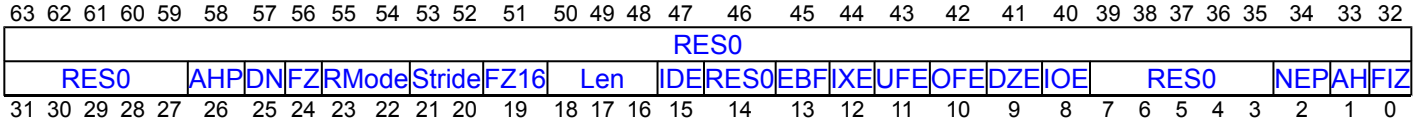
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to FPCR are UNDEFINED.

It is IMPLEMENTATION DEFINED whether the Len and Stride fields can be programmed to nonzero values, which will cause some AArch32 floating-point instruction encodings to be UNDEFINED, or whether these fields are RAZ.

Attributes

FPCR is a 64-bit register.

Field descriptions



Bits [63:27]

Reserved, RES0.

AHP, bit [26]

Alternative half-precision control bit.

AHP	Meaning
0b0	IEEE half-precision format selected.
0b1	Alternative half-precision format selected.

This bit is used only for conversions between half-precision floating-point and other floating-point formats.

The data-processing instructions added as part of the FEAT_FP16 extension always use the IEEE half-precision format, and ignore the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DN, bit [25]

Default NaN use for NaN propagation.

DN	Meaning
0b0	NaN operands propagate through to the output of a floating-point operation.
0b1	Any operation involving one or more NaNs returns the Default NaN. This bit has no effect on the output of the FABS and FNEG instructions. This bit has no effect on the output of the FMAX, FMAXP, FMAXV, FMIN, FMINP, and FMINV instructions when FPCR.AH is 1.

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FZ, bit [24]

Flushing denormalized numbers to zero control bit.

FZ	Meaning
0b0	If FPCR.AH is 0, the flushing to zero of single-precision and double-precision denormalized inputs to, and outputs of, floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero. If FPCR.AH is 1, the flushing to zero of single-precision and double-precision denormalized outputs of floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero.
0b1	If FPCR.AH is 0, denormalized single-precision and double-precision inputs to, and outputs from, floating-point instructions are flushed to zero. If FPCR.AH is 1, denormalized single-precision and double-precision outputs from floating-point instructions are flushed to zero.

For more information, see 'Flushing denormalized numbers to zero' and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RMode, bits [23:22]

Rounding Mode control field.

RMode	Meaning
0b00	Round to Nearest (RN) mode.
0b01	Round towards Plus Infinity (RP) mode.
0b10	Round towards Minus Infinity (RM) mode.
0b11	Round towards Zero (RZ) mode.

The specified rounding mode is used by both scalar and Advanced SIMD floating-point instructions.

If FPCR.AH is 1, then the following instructions use Round to Nearest mode regardless of the value of this bit:

- The FRECPE, FRECPS, FRECPX, FRSQRTE, and FRSQRTS instructions.
- The BFCVT, BFCVTN, BFCVTN2, BFCVTNT, BFMLALB, and BFMLALT instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Stride, bits [21:20]

This field has no function in AArch64 state, and nonzero values are ignored during execution in AArch64 state.

This field is included only for context saving and restoration of the AArch32 FPSCR.Stride field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation implements FPSCR.LEN,STRIDE as RAZ, access to this field is **RAZ/WI**.

FZ16, bit [19]

When FEAT_FP16 is implemented:

Flushing denormalized numbers to zero control bit on half-precision data-processing instructions.

FZ16	Meaning
0b0	For some instructions, this bit disables flushing to zero of inputs and outputs that are half-precision denormalized numbers.
0b1	Flushing denormalized numbers to zero enabled. For some instructions that do not convert a half-precision input to a higher precision output, this bit enables flushing to zero of inputs and outputs that are half-precision denormalized numbers.

The value of this bit applies to both scalar and Advanced SIMD floating-point half-precision calculations.

For more information, see 'Flushing denormalized numbers to zero' and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Len, bits [18:16]

This field has no function in AArch64 state, and nonzero values are ignored during execution in AArch64 state.

This field is included only for context saving and restoration of the AArch32 FPSCR.Len field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation implements FPSCR.LEN,STRIDE as RAZ, access to this field is **RAZ/WI**.

IDE, bit [15]

Input Denormal floating-point exception trap enable.

IDE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.IDC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IDC bit.

When the PE is in Streaming SVE mode, and FEAT_SME_FA64 is not implemented or not enabled, the value of FPCR.IDE is treated as 0 for all purposes other than a direct read or write of the FPCR.

The Effective value of this bit controls both scalar and vector floating-point arithmetic.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Input Denormal floating-point exceptions, access to this field is **RAZ/WI**.

Bit [14]

Reserved, RES0.

EBF, bit [13]

When FEAT_EBF16 is implemented:

The value of this bit controls the numeric behaviors of BFloat16 dot product calculations performed by the BFDOT, BFMMLA, BFMOPA, and BFMOPS instructions. If FEAT_SME2 is implemented, this also controls BFVDOT instruction.

When ID_AA64ISAR1_EL1.BF16 and ID_AA64ZFR0_EL1.BF16 are `0b0010`, the PE supports the FPCR.EBF field. Otherwise, FPCR.EBF is RES0.

EBF	Meaning
`0b0`	These instructions use the standard BFloat16 behaviors: • Ignoring the FPCR.RMode control and using the rounding mode defined for BFloat16. For more information, see 'Round to Odd mode'. • Flushing denormalized inputs and outputs to zero, as if the FPCR.FZ and FPCR.FIZ controls had the value '1'. • Performing unfused multiplies and additions with intermediate rounding of all products and sums.
`0b1`	These instructions use the extended BFloat16 behaviors: • Supporting all four IEEE 754 rounding modes selected by the FPCR.RMode control. • Optionally, flushing denormalized inputs and outputs to zero, as governed by the FPCR.FZ and FPCR.FIZ controls. • Performing a fused two-way sum-of-products for each pair of adjacent BFloat16 elements, without intermediate rounding of the products, but rounding the single-precision sum before addition to the accumulator. • Generating the default NaN as intermediate sum-of-products when any multiplier input is a NaN, or any product is infinity × 0.0, or there are infinite products with differing signs. • Generating an intermediate sum-of-products of the same infinity when there are infinite products all with the same sign.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IXE, bit [12]

Inexact floating-point exception trap enable.

IXE	Meaning
`0b0`	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.IXC bit is set to 1.
`0b1`	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IXC bit.

When the PE is in Streaming SVE mode, and FEAT_SME_FA64 is not implemented or not enabled, the value of FPCR.IXE is treated as 0 for all purposes other than a direct read or write of the FPCR.

The Effective value of this bit controls both scalar and vector floating-point arithmetic.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Inexact floating-point exceptions, access to this field is **RAZ/WI**.

UFE, bit [11]

Underflow floating-point exception trap enable.

UFE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.UFC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs and Flush-to-zero is not enabled, the PE does not update the FPSR.UFC bit.

When the PE is in Streaming SVE mode, and FEAT_SME_FA64 is not implemented or not enabled, the value of FPCR.UFE is treated as 0 for all purposes other than a direct read or write of the FPCR.

The Effective value of this bit controls both scalar and vector floating-point arithmetic.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Underflow floating-point exceptions, access to this field is **RAZ/WI**.

OFE, bit [10]

Overflow floating-point exception trap enable.

OFE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.OFC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.OFC bit.

When the PE is in Streaming SVE mode, and FEAT_SME_FA64 is not implemented or not enabled, the value of FPCR.OFE is treated as 0 for all purposes other than a direct read or write of the FPCR.

The Effective value of this bit controls both scalar and vector floating-point arithmetic.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Overflow floating-point exceptions, access to this field is **RAZ/WI**.

DZE, bit [9]

Divide by Zero floating-point exception trap enable.

DZE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.DZC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.DZC bit.

When the PE is in Streaming SVE mode, and FEAT_SME_FA64 is not implemented or not enabled, the value of FPCR.DZE is treated as 0 for all purposes other than a direct read or write of the FPCR.

The Effective value of this bit controls both scalar and vector floating-point arithmetic.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Divide by Zero floating-point exceptions, access to this field is **RAZ/WI**.

IOE, bit [8]

Invalid Operation floating-point exception trap enable.

IOE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.IOC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IOC bit.

When the PE is in Streaming SVE mode, and FEAT_SME_FA64 is not implemented or not enabled, the value of FPCR.IOE is treated as 0 for all purposes other than a direct read or write of the FPCR.

The Effective value of this bit controls both scalar and vector floating-point arithmetic.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Invalid Operation floating-point exceptions, access to this field is **RAZ/WI**.

Bits [7:3]

Reserved, RES0.

NEP, bit [2]

When FEAT_AFP is implemented:

Controls how the output elements other than the lowest element of the vector are determined for Advanced SIMD scalar instructions.

NEP	Meaning
0b0	Does not affect how the output elements other than the lowest are determined for Advanced SIMD scalar instructions.
0b1	The output elements other than the lowest are taken from the following registers: • For 3-input scalar versions of the FMLA (by element) and FMLS (by element) instructions, the <Hd>, <Sd>, or <Dd> register. • For 3-input versions of the FMADD, FMSUB, FNMADD, and FNMSUB instructions, the <Ha>, <Sa>, or <Da> register. • For 2-input scalar versions of the FACGE, FACGT, FCMEQ (register), FCMGE (register), and FCMGT (register) instructions, the <Hm>, <Sm>, or <Dm> register. • For 2-input scalar versions of the FABD, FADD (scalar), FDIV (scalar), FMAX (scalar), FMAXNM (scalar), FMIN (scalar), FMINNM (scalar), FMUL (by element), FMUL (scalar), FMULX (by element), FMULX, FNMUL (scalar), FRECPS, FRSQRTS, and FSUB (scalar) instructions, the <Hn>, <Sn>, or <Dn> register. • For 1-input scalar versions of the following instructions, the <Hd>, <Sd>, or <Dd> register: ◦ The (vector) versions of the FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, and FCVTPU instructions. ◦ The (vector, fixed-point) and (vector, integer) versions of the FCVTZS, FCVTZU, SCVTF, and UCVTF instructions. ◦ The (scalar) versions of the FABS, FNEG, FRINT32X, FRINT32Z, FRINT64X, FRINT64Z, FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, and FSQRT instructions. ◦ The (scalar, fixed-point) and (scalar, integer) versions of the SCVTF and UCVTF instructions. ◦ The BFCVT, FCVT, FCVTXN, FRECPE, FRECPX, and FRSQRTE instructions.

When the PE is in Streaming SVE mode, and FEAT_SME_FA64 is not implemented or not enabled, the value of FPCR.NEP is treated as 0 for all purposes other than a direct read or write of the FPCR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AH, bit [1]
When FEAT_AFP is implemented:

Alternate Handling. Controls alternate handling of floating-point numbers.

The Arm architecture supports two models for handling some of the corner cases of the floating-point behaviors, such as the nature of flushing of denormalized numbers, the detection of tininess and other exceptions and a range of other behaviors. The value of the FPCR.AH bit selects between these models.

For more information on the FPCR.AH bit, see 'Flushing denormalized numbers to zero', 'Floating-point exceptions and exception traps' and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FIZ, bit [0]
When FEAT_AFP is implemented:

Flush Inputs to Zero. Controls whether single-precision, double-precision and BFloat16 input operands that are denormalized numbers are flushed to zero.

FIZ	Meaning
0b0	The flushing to zero of single-precision and double-precision denormalized inputs to floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero.
0b1	Denormalized single-precision and double-precision inputs to most floating-point instructions flushed to zero.

For more information, see 'Flushing denormalized numbers to zero' and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing FPCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, FPCR
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x00);
        else
            AArch64.SystemAccessTrap(EL1, 0x07);
    elsif ELIsInHost(EL0) && CPTR_EL2.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPCR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif CPACR_EL1.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPCR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPCR;
```

```
MSR FPCR, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x00);
        else
            AArch64.SystemAccessTrap(EL1, 0x07);
    elsif ELIsInHost(EL0) && CPTR_EL2.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPCR = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif CPACR_EL1.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPCR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPCR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPCR = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FPEXC32_EL2, Floating-Point Exception Control Register

The FPEXC32_EL2 characteristics are:

Purpose

Allows access to the AArch32 register FPEXC from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register FPEXC32_EL2 bits [31:0] are architecturally mapped to AArch32 System register FPEXC[31:0].

This register is present only when FEAT_AA32EL1 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to FPEXC32_EL2 are UNDEFINED.

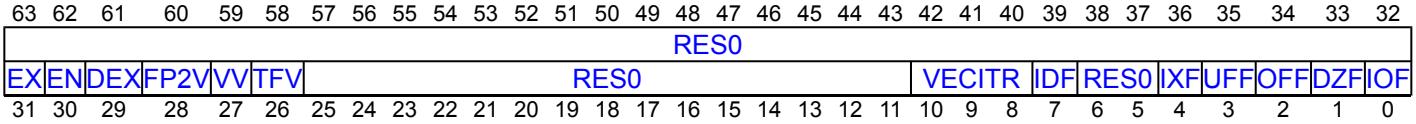
If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

Attributes

FPEXC32_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

EX, bit [31]

Exception bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RAZ/WI**.

EN, bit [30]

Enables access to the Advanced SIMD and floating-point functionality from all Exception levels, except that setting this field to 0 does not disable the following:

- VMSR accesses to the FPEXC or FPSID.
- VMRS accesses from the FPEXC, FPSID, MVFR0, MVFR1, or MVFR2.

EN	Meaning
0b0	Accesses to the FPSCR, and any of the SIMD and floating-point registers Q0-Q15, including their views as D0-D31 registers or S0-S31 registers, are UNDEFINED at all Exception levels.
0b1	This control permits access to the Advanced SIMD and floating-point functionality at all Exception levels.

Execution of Advanced SIMD and floating-point instructions in AArch32 state can be disabled or trapped by the following controls:

- CPACR.cp10, or, if executing at EL0, CPACR_EL1.FPEN.
- FPEXC.EN.
- If executing in Non-secure state:
  - HCPTR.TCP10, or if EL2 is using AArch64, CPTR_EL2.TFP.
  - NSACR.cp10, or if EL3 is using AArch64, CPTR_EL3.TFP.
- For Advanced SIMD instructions only:
  - CPACR.ASEDIS.
  - If executing in Non-secure state, HCPTR.TASE and NSACR.NSASEDIS.

See the descriptions of the controls for more information.

Note

When executing at EL0 using AArch32:

- If EL1 is using AArch64, then the Effective value of FPEXC.EN is 1.
- If EL2 is using AArch64 and is enabled in the current Security state, HCR_EL2.TGE is 1, and the Effective value of HCR_EL2.RW is 1, then the Effective value of FPEXC.EN is 1. However, Arm deprecates using the value of FPEXC32_EL2.EN to determine behavior.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DEX, bit [29]

Defined synchronous exception on floating-point execution.

This field identifies whether a synchronous exception generated by the attempted execution of an instruction was generated by an unallocated encoding. The instruction must be in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr() returning TRUE. This field also indicates whether the FPEXC32_EL2.TFV field is valid.

The meaning of this bit is:

DEX	Meaning
0b0	The exception was generated by the attempted execution of an unallocated instruction in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr(). If FPEXC32_EL2.TFV is RW then it is invalid and UNKNOWN. If FPEXC32_EL2.{IDF, IXF, UFF, OFF, DZF, IOF} are RW then they are invalid and UNKNOWN.
0b1	The exception was generated during the execution of an allocated encoding. FPEXC32_EL2.TFV is valid and indicates the cause of the exception.

On an exception that sets this bit to 1 the exception-handling routine must clear this bit to 0.

On an implementation that both does not support trapping of floating-point exceptions and implements the AArch32 FPSCR.{Stride, Len} fields as RAZ, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FP2V, bit [28]

FPINST2 instruction Valid bit. From Armv8.0, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RES0**.

VV, bit [27]

VECITR valid bit. From Armv8, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RES0**.

TFV, bit [26]

Trapped Fault Valid bit. Valid only when the value of FPEXC32_EL2.DEX is 1. When valid, it indicates the cause of the exception and therefore whether FPEXC32_EL2.{IDF, IXF, UFF, OFF, DZF, IOF} are valid.

TFV	Meaning
0b0	The exception was caused by the execution of a floating-point VABS, VADD, VDIV, VFMA, VFMS, VFNMA, VFNMS, VMLA, VMLS, VMOV, VMUL, VNEG, VNMLA, VNMLS, VNMUL, VSQRT, or VSUB instruction when one or both of FPSCR.{Stride, Len} was nonzero. If FPEXC32_EL2.{IDF, IXF, UFF, OFF, DZF, IOF} are RW then they are invalid and UNKNOWN.
0b1	FPEXC32_EL2.{IDF, IXF, UFF, OFF, DZF, IOF} indicate the presence of trapped floating-point exceptions that had occurred at the time of the exception. Bits are set for all trapped exceptions that had occurred at the time of the exception.

This bit returns a status value and ignores writes.

When the value of FPEXC32_EL2.DEX is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When an implementation does not implement trapping of floating-point exceptions, access to this field is **RAZ/WI**.
- When an implementation implements FPSCR.LEN,STRIDE as RAZ, access to this field is **RAO/WI**.

Bits [25:11]

Reserved, RES0.

VECITR, bits [10:8]

Vector iteration count. From Armv8, this field is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RES1**.

IDF, bit [7]

Input Denormal trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Input Denormal exception occurred while FPSCR.IDE was 1:

IDF	Meaning
0b0	Input Denormal exception has not occurred.
0b1	Input Denormal exception has occurred.

Input Denormal exceptions can occur only when FPSCR.FZ is 1.

Note

A half-precision floating-point value that is flushed to zero because the value of FPSCR.FZ16 is 1 does not generate an Input Denormal exception.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC32_EL2.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Input Denormal floating-point exceptions, access to this field is **RAZ/WI**.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Inexact exception occurred while FPSCR.IXE was 1:

IXF	Meaning
0b0	Inexact exception has not occurred.
0b1	Inexact exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Inexact floating-point exceptions, access to this field is **RAZ/WI**.

UFF, bit [3]

Underflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Underflow exception occurred while FPSCR.UFE was 1:

UFF	Meaning
0b0	Underflow exception has not occurred.
0b1	Underflow exception has occurred.

Underflow trapped exceptions can occur:

- On half-precision data-processing instructions only when FPSCR.FZ16 is 0.
- Otherwise only when FPSCR.FZ is 0.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC32_EL2.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Underflow floating-point exceptions, access to this field is **RAZ/WI**.

OFF, bit [2]

Overflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Overflow exception occurred while FPSCR.OFE was 1:

OFF	Meaning
0b0	Overflow exception has not occurred.
0b1	Overflow exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Overflow floating-point exceptions, access to this field is **RAZ/WI**.

DZF, bit [1]

Divide by Zero trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether a Divide by Zero exception occurred while FPSCR.DZE was 1:

DZF	Meaning
0b0	Divide by Zero exception has not occurred.
0b1	Divide by Zero exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Divide by Zero floating-point exceptions, access to this field is **RAZ/WI**.

IOF, bit [0]

Invalid Operation trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Invalid Operation exception occurred while FPSCR.IOE was 1:

IOF	Meaning
0b0	Invalid Operation exception has not occurred.
0b1	Invalid Operation exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Invalid Operation floating-point exceptions, access to this field is **RAZ/WI**.

Accessing FPEXC32_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, FPEXC32_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPEXC32_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPEXC32_EL2;
```

```
MSR FPEXC32_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPEXC32_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPEXC32_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FPMR, Floating-point Mode Register

The FPMR characteristics are:

Purpose

Controls behaviors of the FP8 instructions.

Configuration

This register is present only when FEAT_FPMR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to FPMR are UNDEFINED.

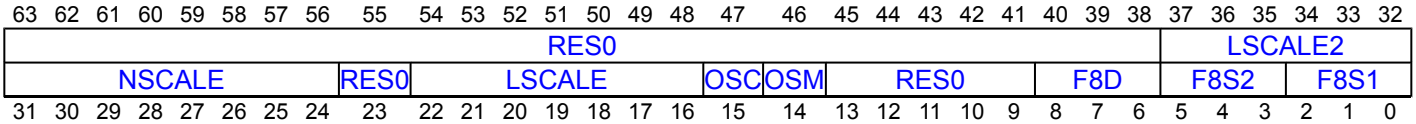
A direct or indirect read of this register occurs in program order relative to a direct write of this register without explicit synchronization.

On entry to or exit from Streaming SVE mode, FPMR is set to 0.

Attributes

FPMR is a 64-bit register.

Field descriptions



Bits [63:38]

Reserved, RES0.

LSCALE2, bits [37:32]

Downscaling value for instructions that convert the second FP8 input data stream to other floating-point formats.

This value is an unsigned integer that is subtracted from the result exponent.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSCALE, bits [31:24]

Scaling value for instructions that convert other floating-point formats to an FP8 format.

This value is a signed integer that is added to the operand exponent.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [23]

Reserved, RES0.

LSCALE, bits [22:16]

Downscaling value.

This value is an unsigned integer that is subtracted from:

- The product or the sum-of-products exponent, for multiplication instructions with FP8 operands.
- The result exponent, for instructions that convert the first FP8 input data stream to other floating-point formats.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OSC, bit [15]

Overflow saturation for FP8 convert instructions. Specifies the result when a floating-point Overflow exception is detected.

OSC	Meaning
0b0	Infinity or NaN is generated.
0b1	Maximum normal number is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OSM, bit [14]

Overflow saturation for FP8 multiplication instructions. Specifies the result when a floating-point Overflow exception is detected.

OSM	Meaning
0b0	Infinity is generated.
0b1	Maximum normal number is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [13:9]

Reserved, RES0.

F8D, bits [8:6]

Destination result format for instructions that convert other floating-point values to an FP8 format.

F8D	Meaning
0b000	OFP8 E5M2 format.
0b001	OFP8 E4M3 format.

All other values are reserved.

Reserved values identify an unsupported format, causing convert instructions that generate an FP8 result to perform a CONSTRAINED UNPREDICTABLE choice of one of the following behaviors:

- Setting the result to 0xFF and signaling an Invalid Operation floating-point exception.
- Generating the expected result of any of the supported FP8 formats.

It is software's responsibility to check that a format value is supported in ID_AA64FPFR0_EL1[7:0], before writing it to this field.

For more information about the FP8 formats, see the OCP 8-bit Floating Point Specification (OFP8).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F8S2, bits [5:3]

Second FP8 input data stream format for multiplication instructions with FP8 operands, and the corresponding instructions that convert an FP8 format to other floating-point formats.

F8S2	Meaning
0b000	OFP8 E5M2 format.
0b001	OFP8 E4M3 format.

All other values are reserved.

Reserved values identify an unsupported format, and FP8 instructions treat the corresponding input as a CONSTRAINED UNPREDICTABLE choice of one of the following:

- A signaling NaN.
- Any of the supported FP8 formats.

It is software's responsibility to check that a format value is supported in ID_AA64FPFR0_EL1[7:0], before writing it to this field.

For more information about the FP8 formats, see the OCP 8-bit Floating Point Specification (OFP8).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F8S1, bits [2:0]

First FP8 input data stream format for multiplication instructions with FP8 operands, and the corresponding instructions that convert an FP8 format to other floating-point formats.

F8S1	Meaning
0b000	OFP8 E5M2 format.
0b001	OFP8 E4M3 format.

All other values are reserved.

Reserved values identify an unsupported format, and FP8 instructions treat the corresponding input as a CONSTRAINED UNPREDICTABLE choice of one of the following:

- A signaling NaN.
- Any of the supported FP8 formats.

It is software's responsibility to check that a format value is supported in ID_AA64FPFR0_EL1[7:0], before writing it to this field.

For more information about the FP8 formats, see the OCP 8-bit Floating Point Specification (OFP8).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FPMR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, FPMR
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_FPMR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnFPM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.EnFPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnFPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && ((HaveEL(EL3) && SCR_EL3.HXEn == '0') ||
HCRX_EL2.EnFPM == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnFPM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !ELIsInHost(EL0) && CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x00);
        else
            AArch64.SystemAccessTrap(EL1, 0x07);
    elsif ELIsInHost(EL0) && CPTR_EL2.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPMR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnFPM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELIsInHost(EL0) && ((HaveEL(EL3) && SCR_EL3.HXEn == '0') ||
HCRX_EL2.EnFPM == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnFPM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif CPACR_EL1.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPMR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnFPM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
```

```
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnFPM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPMR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPMR;
```

```
MSR FPMR, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_FPMR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnFPM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.EnFPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnFPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && ((HaveEL(EL3) && SCR_EL3.HXEn == '0') ||
HCRX_EL2.EnFPM == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnFPM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !ELIsInHost(EL0) && CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x00);
        else
            AArch64.SystemAccessTrap(EL1, 0x07);
    elsif ELIsInHost(EL0) && CPTR_EL2.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPMR = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnFPM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELIsInHost(EL0) && ((HaveEL(EL3) && SCR_EL3.HXEn == '0') ||
HCRX_EL2.EnFPM == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnFPM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif CPACR_EL1.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPMR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnFPM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
```

```
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnFPM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPMR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPMR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FPSR, Floating-point Status Register

The FPSR characteristics are:

Purpose

Provides floating-point system status information.

Configuration

AArch64 System register FPSR bits [31:27] are architecturally mapped to AArch32 System register FPSCR[31:27].

AArch64 System register FPSR bit [7] is architecturally mapped to AArch32 System register FPSCR[7].

AArch64 System register FPSR bits [4:0] are architecturally mapped to AArch32 System register FPSCR[4:0].

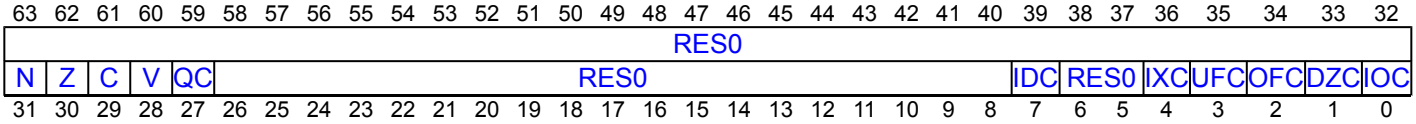
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to FPSR are UNDEFINED.

On entry to or exit from Streaming SVE mode, FPSR.{IOC, DZC, OFC, UFC, IXC, IDC, QC} are set to 1 and the remaining bits are set to 0.

Attributes

FPSR is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

N, bit [31]

When FEAT_AA32 is implemented and FEAT_FP is implemented:

Negative condition flag for AArch32 floating-point comparison operations.

Note

AArch64 floating-point comparisons set the PSTATE.N flag instead.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Z, bit [30]

When FEAT_AA32 is implemented and FEAT_FP is implemented:

Zero condition flag for AArch32 floating-point comparison operations.

Note

AArch64 floating-point comparisons set the PSTATE.Z flag instead.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

C, bit [29]

When FEAT_AA32 is implemented and FEAT_FP is implemented:

Carry condition flag for AArch32 floating-point comparison operations.

Note

AArch64 floating-point comparisons set the PSTATE.C flag instead.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

V, bit [28]

When FEAT_AA32 is implemented and FEAT_FP is implemented:

Overflow condition flag for AArch32 floating-point comparison operations.

Note

AArch64 floating-point comparisons set the PSTATE.V flag instead.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

QC, bit [27]

Cumulative saturation bit, Advanced SIMD only. This bit is set to 1 to indicate that an Advanced SIMD integer operation has saturated since 0 was last written to this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [26:8]

Reserved, RES0.

IDC, bit [7]

Input Denormal cumulative floating-point exception bit. This bit is set to 1 to indicate that the Input Denormal floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.IDE bit. This bit is set to 1 to indicate a floating-point exception only if FPCR.IDE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXC, bit [4]

Inexact cumulative floating-point exception bit. This bit is set to 1 to indicate that the Inexact floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.IXE bit. This bit is set to 1 to indicate a floating-point exception only if FPCR.IXE is 0.

The criteria for the Inexact floating-point exception to occur are affected by whether denormalized numbers are flushed to zero and by the value of the FPCR.AH bit. For more information, see 'Floating-point exceptions and exception traps'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFC, bit [3]

Underflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Underflow floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.UFE bit. This bit is set to 1 to indicate a floating-point exception only if FPCR.UFE is 0 or if flushing denormalized numbers to zero is enabled.

The criteria for the Underflow floating-point exception to occur are affected by whether denormalized numbers are flushed to zero and by the value of the FPCR.AH bit. For more information, see 'Floating-point exceptions and exception traps'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFC, bit [2]

Overflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Overflow floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.OFE bit. This bit is set to 1 to indicate a floating-point exception only if FPCR.OFE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZC, bit [1]

Divide by Zero cumulative floating-point exception bit. This bit is set to 1 to indicate that the Divide by Zero floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.DZE bit. This bit is set to 1 to indicate a floating-point exception only if FPCR.DZE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOC, bit [0]

Invalid Operation cumulative floating-point exception bit. This bit is set to 1 to indicate that the Invalid Operation floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.IOE bit. This bit is set to 1 to indicate a floating-point exception only if FPCR.IOE is 0.

The criteria for the Invalid Operation floating-point exception to occur are affected by the value of the FPCR.AH bit. For more information, see 'Floating-point exceptions and exception traps'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FPSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, FPSR
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x00);
        else
            AArch64.SystemAccessTrap(EL1, 0x07);
    elsif ELIsInHost(EL0) && CPTR_EL2.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPSR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif CPACR_EL1.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPSR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        X[t, 64] = FPSR;
```

```
MSR FPSR, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x00);
        else
            AArch64.SystemAccessTrap(EL1, 0x07);
    elsif ELIsInHost(EL0) && CPTR_EL2.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPSR = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif CPACR_EL1.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPSR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && CPTR_EL2.FPEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPSR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPSR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCR_EL1, Tag Control Register.

The GCR_EL1 characteristics are:

Purpose

Tag Control Register.

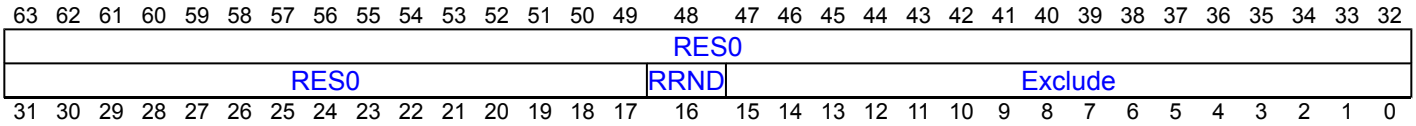
Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to GCR_EL1 are UNDEFINED.

Attributes

GCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:17]

Reserved, RES0.

RRND, bit [16]

Controls generation of tag values by the `IRG` instruction.

RRND	Meaning
`0b0`	`IRG` generates a tag value as defined by RandomTag() and ChooseNonExcludedTag(). This mode does not provide strong guarantees for randomness and should only be used for debugging purposes.
`0b1`	`IRG` generates an implementation-specific tag value with a distribution of tag values no worse than generated with GCR_EL1.RRND == 0.

Note

Arm recommends that IMPLEMENTATION DEFINED algorithms minimize the risk of a bias by selecting tags from a uniform distribution.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Exclude, bits [15:0]

Allocation Tag values excluded from selection by ChooseNonExcludedTag().

If all bits of GCR_EL1.Exclude are 1, then the Allocation Tag value 0 will be used.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b110

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCR_EL1;
```

```
MSR GCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b110

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSCR_EL1, Guarded Control Stack Control Register (EL1)

The GCSCR_EL1 characteristics are:

Purpose

Controls the Guarded Control Stack at EL1.

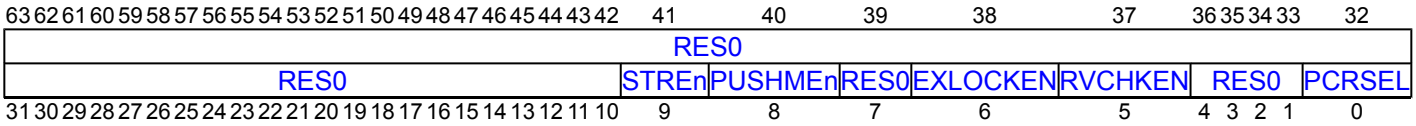
Configuration

This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSCR_EL1 are UNDEFINED.

Attributes

GCSCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:10]

Reserved, RES0.

STREn, bit [9]

Execution of the following instructions are trapped:

- GCSSTR.
- GCSSTTR if any of the following are true.
  - PSTATE.UAO is 1.
  - If EL2 is implemented and enabled in the current Security state and the Effective value of HCR_EL2.{NV,NV1} is {1,1}.

STREn	Meaning
0b0	Execution of any of the specified instructions at EL1 causes a GCS exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

PUSHMEn, bit [8]

Trap GCSPUSHM instruction.

PUSHMEn	Meaning
0b0	Execution of a GCSPUSHM instruction at EL1 causes a Trap exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [7]

Reserved, RES0.

EXLOCKEN, bit [6]

Exception state lock.

Prevents MSR instructions from writing to ELR_EL1 or SPSR_EL1.

EXLOCKEN	Meaning
0b0	EL1 exception state locking disabled.
0b1	EL1 exception state locking enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

RVCHKEN, bit [5]

Return value check enable.

RVCHKEN	Meaning
0b0	Return value checking disabled at EL1.
0b1	Return value checking enabled at EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:1]

Reserved, RES0.

PCRSEL, bit [0]

Guarded Control Stack procedure call return enable selection.

PCRSEL	Meaning
0b0	Guarded Control Stack at EL1 is not PCR Selected.
0b1	Guarded Control Stack at EL1 is PCR Selected.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing GCSCR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name GCSCR_EL1 or GCSCR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8D0];
    else
        X[t, 64] = GCSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = GCSCR_EL2;
    else
        X[t, 64] = GCSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSCR_EL1;
```

```
MSR GCSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8D0] = X[t, 64];
    else
        GCSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        GCSCR_EL2 = X[t, 64];
    else
        GCSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, GCSCR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x8D0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = GCSCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = GCSCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR GCSCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x8D0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            GCSCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        GCSCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSCR_EL2, Guarded Control Stack Control Register (EL2)

The GCSCR_EL2 characteristics are:

Purpose

Controls the Guarded Control Stack at EL2.

Configuration

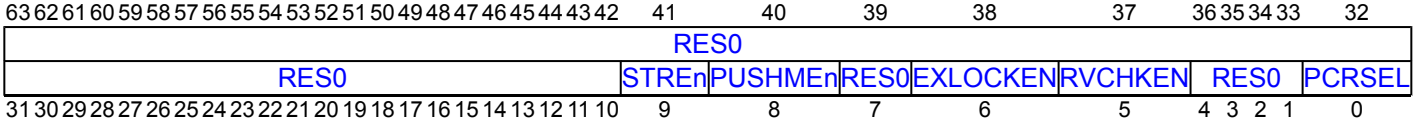
This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSCR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

GCSCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:10]

Reserved, RES0.

STREn, bit [9]

Execution of the following instructions are trapped:

- GCSSTR.
- GCSSTTR if any of the following are true.
  - The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
  - The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, and PSTATE.UAO is 1.

STREn	Meaning
0b0	Execution of any of the specified instructions at EL2 cause a GCS exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

PUSHMEn, bit [8]

Trap GCSPUSHM instruction.

PUSHMEn	Meaning
0b0	Execution of a GCSPUSHM instruction at EL2 causes a Trap exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [7]

Reserved, RES0.

EXLOCKEN, bit [6]

Exception state lock.

Prevents MSR instructions from writing to ELR_EL2 or SPSR_EL2.

EXLOCKEN	Meaning
0b0	EL2 exception state locking disabled.
0b1	EL2 exception state locking enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

RVCHKEN, bit [5]

Return value check enable.

RVCHKEN	Meaning
0b0	Return value checking disabled at EL2.
0b1	Return value checking enabled at EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:1]

Reserved, RES0.

PCRSEL, bit [0]

Guarded Control Stack procedure call return enable selection.

PCRSEL	Meaning
0b0	Guarded Control Stack at EL2 is not PCR Selected.
0b1	Guarded Control Stack at EL2 is PCR Selected.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing GCSCR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name GCSCR_EL2 or GCSCR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSCR_EL2;
```

MSR GCSCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSCR_EL2 = X[t, 64];
```

MRS <Xt>, GCSCR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8D0];
    else
        X[t, 64] = GCSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = GCSCR_EL2;
    else
        X[t, 64] = GCSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSCR_EL1;
```

```
MSR GCSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8D0] = X[t, 64];
    else
        GCSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        GCSCR_EL2 = X[t, 64];
    else
        GCSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSCR_EL3, Guarded Control Stack Control Register (EL3)

The GCSCR_EL3 characteristics are:

Purpose

Controls the Guarded Control Stack at EL3.

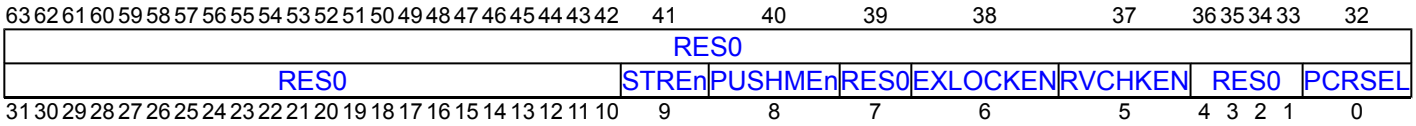
Configuration

This register is present only when FEAT_GCS is implemented and EL3 is implemented. Otherwise, direct accesses to GCSCR_EL3 are UNDEFINED.

Attributes

GCSCR_EL3 is a 64-bit register.

Field descriptions



Bits [63:10]

Reserved, RES0.

STREn, bit [9]

Execution of the following instructions are trapped:

- GCSSTR.
- GCSSTTR.

STREn	Meaning
0b0	Execution of any of the specified instructions at EL3 cause a GCS exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

PUSHMEn, bit [8]

Trap GCSPUSHM instruction.

PUSHMEn	Meaning
0b0	Execution of a GCSPUSHM instruction at EL3 causes a Trap exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [7]

Reserved, RES0.

EXLOCKEN, bit [6]

Exception state lock.

Prevents MSR instructions from writing to ELR_EL3 or SPSR_EL3.

EXLOCKEN	Meaning
0b0	EL3 exception state locking disabled.
0b1	EL3 exception state locking enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

RVCHKEN, bit [5]

Return value check enable.

RVCHKEN	Meaning
0b0	Return value checking disabled at EL3.
0b1	Return value checking enabled at EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:1]

Reserved, RES0.

PCRSEL, bit [0]

Guarded Control Stack procedure call return enable selection.

PCRSEL	Meaning
0b0	Guarded Control Stack at EL3 is not PCR Selected.
0b1	Guarded Control Stack at EL3 is PCR Selected.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing GCSCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0101	0b000

```
if !(IsFeatureImplemented(FEAT_GCS) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSCR_EL3;
```

```
MSR GCSCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0101	0b000

```
if !(IsFeatureImplemented(FEAT_GCS) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.GCSCR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSCR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSCRE0_EL1, Guarded Control Stack Control Register (EL0)

The GCSCRE0_EL1 characteristics are:

Purpose

Controls the Guarded Control Stack at EL0.

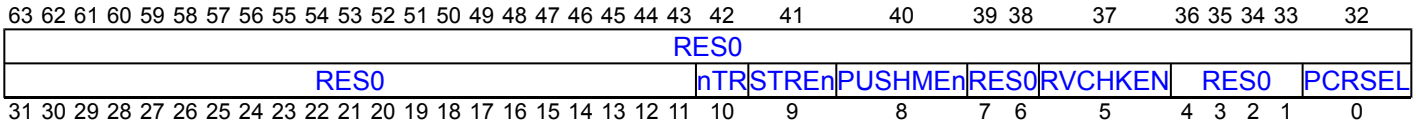
Configuration

This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSCRE0_EL1 are UNDEFINED.

Attributes

GCSCRE0_EL1 is a 64-bit register.

Field descriptions



Bits [63:11]

Reserved, RES0.

nTR, bit [10]

Trap GCS register accesses from EL0.

nTR	Meaning
0b0	Read accesses to GCSPR_EL0 at EL0 cause a Trap exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

STREn, bit [9]

Execution of the following instructions are trapped:

- GCSSTR.
- GCSSTTR.

STREn	Meaning
0b0	Execution of any of the specified instructions at EL0 cause a GCS exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

PUSHMEn, bit [8]

Trap GCSPUSHM instruction.

PUSHMEn	Meaning
0b0	Execution of a GCSPUSHM instruction at EL0 causes a Trap exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [7:6]

Reserved, RES0.

RVCHKEN, bit [5]

Return value check enable.

RVCHKEN	Meaning
0b0	Return value checking disabled at EL0.
0b1	Return value checking enabled at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:1]

Reserved, RES0.

PCRSEL, bit [0]

Guarded Control Stack procedure call return enable selection.

PCRSEL	Meaning
0b0	Guarded Control Stack at EL0 is not PCR Selected.
0b1	Guarded Control Stack at EL0 is PCR Selected.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing GCSCRE0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSCRE0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b010

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nGCS_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSCRE0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSCRE0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSCRE0_EL1;
```

```
MSR GCSCRE0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b010

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nGCS_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSCRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSCRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSCRE0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSPOPCX, Guarded Control Stack Pop and Compare exception return record

The GCSPOPCX characteristics are:

Purpose

Loads an exception return record from the location indicated by the current Guarded Control Stack Pointer register, compares the values loaded with the current ELR_ELx, SPSR_ELx, and LR, and increments the current Guarded Control Stack Pointer register by the size of a Guarded Control Stack exception return record.

Configuration

This instruction is present only when FEAT_GCS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to GCSPOPCX are UNDEFINED.

Attributes

GCSPOPCX is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing GCSPOPCX

Rt should be encoded as `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

`GCSPOPCX`

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0111	0b101

```
if !(IsFeatureImplemented(FEAT_GCS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
then
        EXLOCKException();
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.nGCSEPP == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif GCSEnabled(EL1) then
        GCSPOPCX();
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
then
        EXLOCKException();
    elsif GCSEnabled(EL2) then
        GCSPOPCX();
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
then
        EXLOCKException();
    elsif GCSEnabled(EL3) then
        GCSPOPCX();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSPOPM, Guarded Control Stack Pop

The GCSPOPM characteristics are:

Purpose

Loads the 64-bit doubleword that is pointed to by the current Guarded Control Stack Pointer, writes it to the destination register, and increments the current Guarded Control Stack Pointer register by the size of a Guarded Control Stack procedure return record.

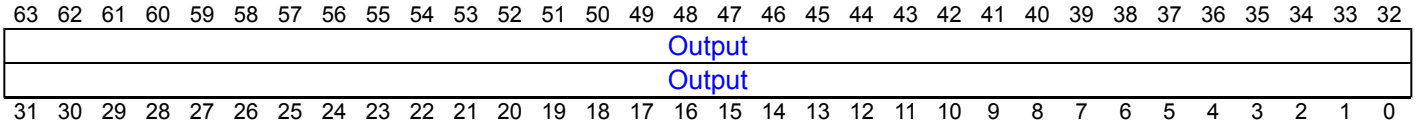
Configuration

This instruction is present only when FEAT_GCS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to GCSPOPM are UNDEFINED.

Attributes

GCSPOPM is a 64-bit System instruction.

Field descriptions



Output, bits [63:0]

Output value for Guarded Control Stack procedure return record.

Executing GCSPOPM

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
GCSPOPM {<Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0111	0b001

```
if !(IsFeatureImplemented(FEAT_GCS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif GCSEnabled(PSTATE.EL) then
    X[t, 64] = GCSPOPM();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSPOPX, Guarded Control Stack Pop exception return record

The GCSPOPX characteristics are:

Purpose

Loads an exception return record from the location indicated by the current Guarded Control Stack Pointer register, checks that the record is a Guarded Control Stack exception return record, and increments the current Guarded Control Stack Pointer register by the size of a Guarded Control Stack exception return record.

Configuration

This instruction is present only when FEAT_GCS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to GCSPOPX are UNDEFINED.

Attributes

GCSPOPX is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing GCSPOPX

Rt should be encoded as `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

`GCSPOPX`

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0111	0b110

```
if !(IsFeatureImplemented(FEAT_GCS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if GCSEnabled(EL1) then
        GCSPOPX();
elsif PSTATE.EL == EL2 then
    if GCSEnabled(EL2) then
        GCSPOPX();
elsif PSTATE.EL == EL3 then
    if GCSEnabled(EL3) then
        GCSPOPX();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSPR_EL0, Guarded Control Stack Pointer Register (EL0)

The GCSPR_EL0 characteristics are:

Purpose

Contains the Guarded Control Stack Pointer at EL0.

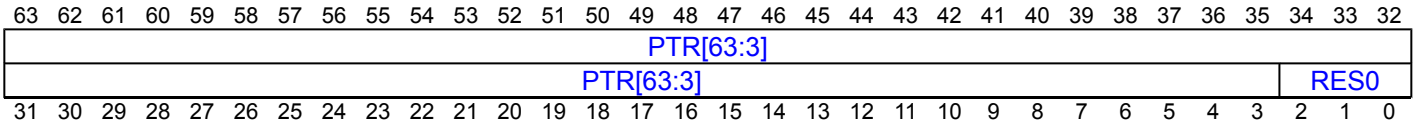
Configuration

This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSPR_EL0 are UNDEFINED.

Attributes

GCSPR_EL0 is a 64-bit register.

Field descriptions



PTR[63:3], bits [63:3]

EL0 Guarded Control Stack Pointer bits [63:3].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [2:0]

Reserved, RES0.

Accessing GCSPR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSPR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif (!EL2Enabled() || HCR_EL2.TGE != '1') && GCSCRE0_EL1.nTR == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.TGE == '1' && GCSCRE0_EL1.nTR == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGRTR_EL2.nGCS_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSPR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nGCS_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSPR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSPR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSPR_EL0;
```

```
MSR GCSPR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nGCS_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSPR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSPR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSPR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSPR_EL1, Guarded Control Stack Pointer Register (EL1)

The GCSPR_EL1 characteristics are:

Purpose

Contains the Guarded Control Stack Pointer at EL1.

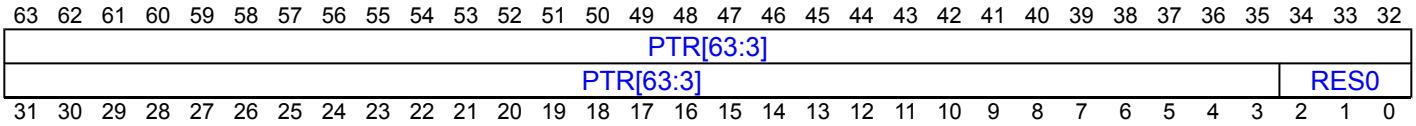
Configuration

This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSPR_EL1 are UNDEFINED.

Attributes

GCSPR_EL1 is a 64-bit register.

Field descriptions



PTR[63:3], bits [63:3]

EL1 Guarded Control Stack Pointer bits [63:3].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [2:0]

Reserved, RES0.

Accessing GCSPR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name GCSPR_EL1 or GCSPR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSPR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8C0];
    else
        X[t, 64] = GCSPR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = GCSPR_EL2;
    else
        X[t, 64] = GCSPR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSPR_EL1;
```

```
MSR GCSPR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8C0] = X[t, 64];
    else
        GCSPR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        GCSPR_EL2 = X[t, 64];
    else
        GCSPR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSPR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, GCSPR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x8C0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = GCSPR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = GCSPR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR GCSPR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x8C0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            GCSPR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        GCSPR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSPR_EL2, Guarded Control Stack Pointer Register (EL2)

The GCSPR_EL2 characteristics are:

Purpose

Contains the Guarded Control Stack Pointer at EL2.

Configuration

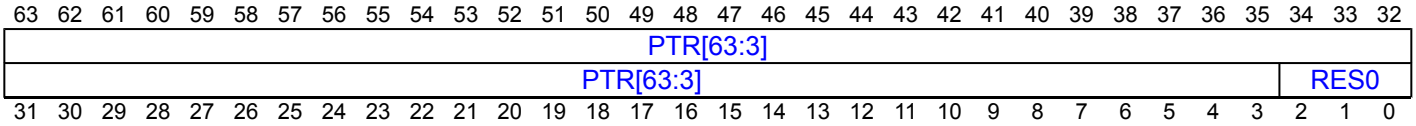
This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSPR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

GCSPR_EL2 is a 64-bit register.

Field descriptions



PTR[63:3], bits [63:3]

EL2 Guarded Control Stack Pointer bits [63:3].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [2:0]

Reserved, RES0.

Accessing GCSPR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name GCSPR_EL2 or GCSPR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSPR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSPR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSPR_EL2;
```

MSR GCSPR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSPR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSPR_EL2 = X[t, 64];
```

MRS <Xt>, GCSPR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8C0];
    else
        X[t, 64] = GCSPR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = GCSPR_EL2;
    else
        X[t, 64] = GCSPR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSPR_EL1;
```

```
MSR GCSPR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8C0] = X[t, 64];
    else
        GCSPR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        GCSPR_EL2 = X[t, 64];
    else
        GCSPR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSPR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSPR_EL3, Guarded Control Stack Pointer Register (EL3)

The GCSPR_EL3 characteristics are:

Purpose

Contains the Guarded Control Stack Pointer at EL3.

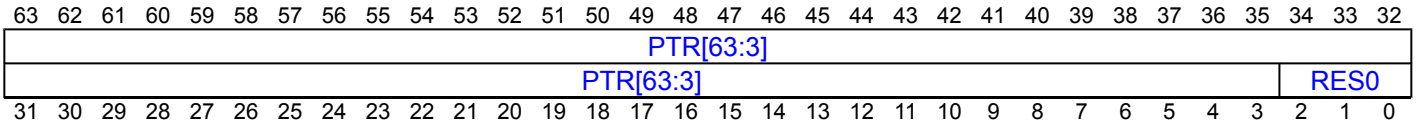
Configuration

This register is present only when FEAT_GCS is implemented and EL3 is implemented. Otherwise, direct accesses to GCSPR_EL3 are UNDEFINED.

Attributes

GCSPR_EL3 is a 64-bit register.

Field descriptions



PTR[63:3], bits [63:3]

EL3 Guarded Control Stack Pointer bits [63:3].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [2:0]

Reserved, RES0.

Accessing GCSPR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSPR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_GCS) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSPR_EL3;
```

```
MSR GCSPR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_GCS) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.GCSPR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSPR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSPUSHM, Guarded Control Stack Push

The GCSPUSHM characteristics are:

Purpose

Decrements the current Guarded Control Stack Pointer register by the size of a Guarded Control Stack procedure return record and stores an entry to the Guarded Control Stack.

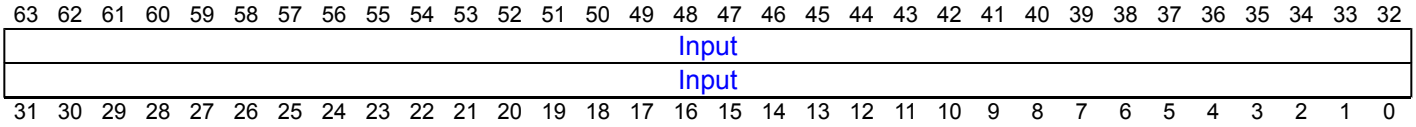
Configuration

This instruction is present only when FEAT_GCS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to GCSPUSHM are UNDEFINED.

Attributes

GCSPUSHM is a 64-bit System instruction.

Field descriptions



Input, bits [63:0]

Input value for Guarded Control Stack procedure return record.

Executing GCSPUSHM

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
GCSPUSHM <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0111	0b000

```
if !(IsFeatureImplemented(FEAT_GCS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if (!EL2Enabled() || HCR_EL2.TGE != '1') && GCSCRE0_EL1.PUSHMEn == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.TGE == '1' && GCSCRE0_EL1.PUSHMEn == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif GCSEnabled(EL0) then
        GCSPUSHM(X[t, 64]);
elsif PSTATE.EL == EL1 then
    if GCSCR_EL1.PUSHMEn == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.nGCSPUSHM_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif GCSEnabled(EL1) then
        GCSPUSHM(X[t, 64]);
elsif PSTATE.EL == EL2 then
    if GCSCR_EL2.PUSHMEn == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif GCSEnabled(EL2) then
        GCSPUSHM(X[t, 64]);
elsif PSTATE.EL == EL3 then
    if GCSCR_EL3.PUSHMEn == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif GCSEnabled(EL3) then
        GCSPUSHM(X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSPUSHX, Guarded Control Stack Push exception return record

The GCSPUSHX characteristics are:

Purpose

Decrements the current Guarded Control Stack Pointer register by the size of a Guarded Control Stack exception return record and stores a Guarded Control Stack exception return record to the Guarded Control Stack.

Configuration

This instruction is present only when FEAT_GCS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to GCSPUSHX are UNDEFINED.

Attributes

GCSPUSHX is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing GCSPUSHX

Rt should be encoded as `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

`GCSPUSHX`

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0111	0b100

```
if !(IsFeatureImplemented(FEAT_GCS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '0'
then
        EXLOCKException();
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.nGCSEPP == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif GCSEnabled(EL1) then
        GCSPUSHX();
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '0'
then
        EXLOCKException();
    elsif GCSEnabled(EL2) then
        GCSPUSHX();
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '0'
then
        EXLOCKException();
    elsif GCSEnabled(EL3) then
        GCSPUSHX();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSSS1, Guarded Control Stack Switch Stack 1

The GCSSS1 characteristics are:

Purpose

Validates that the stack being switched to contains a Valid cap entry, stores an In-progress cap entry on to the stack that is getting switched to and sets the current Guarded Control Stack Pointer to the stack that is getting switched to.

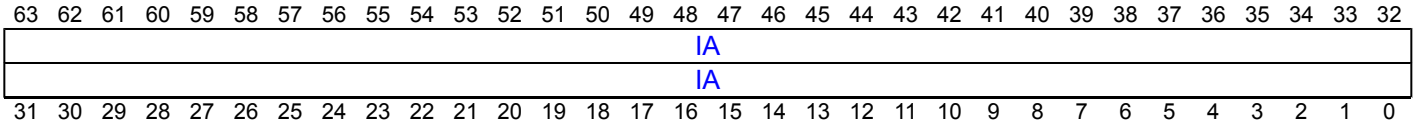
Configuration

This instruction is present only when FEAT_GCS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to GCSSS1 are UNDEFINED.

Attributes

GCSSS1 is a 64-bit System instruction.

Field descriptions



IA, bits [63:0]

Incoming address, for the incoming Guarded Control Stack.

Executing GCSSS1

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
GCSSS1 <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_GCS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif GCSEnabled(PSTATE.EL) then
    GCSSS1(X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GCSSS2, Guarded Control Stack Switch Stack 2

The GCSSS2 characteristics are:

Purpose

Validates that the most recent entry of the Guarded Control Stack that is getting switched to contains an In-progress cap entry, stores a Valid cap entry to the Guarded Control Stack that is getting switched from, and sets Xt to the address of that Valid cap entry.

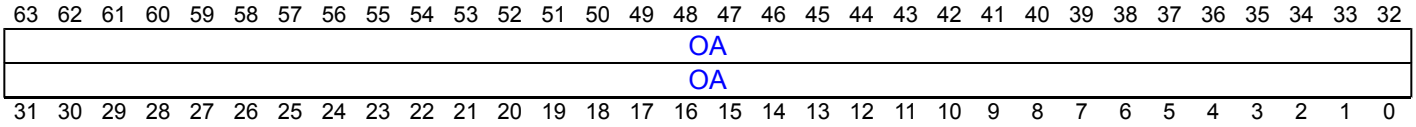
Configuration

This instruction is present only when FEAT_GCS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to GCSSS2 are UNDEFINED.

Attributes

GCSSS2 is a 64-bit System instruction.

Field descriptions



OA, bits [63:0]

Outgoing address, for the outgoing Guarded Control Stack.

Executing GCSSS2

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
GCSSS2 <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_GCS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif GCSEnabled(PSTATE.EL) then
    X[t, 64] = GCSSS2();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GMID_EL1, Multiple tag transfer ID Register

The GMID_EL1 characteristics are:

Purpose

Indicates the block size that is accessed by the LDGM and STGM System instructions.

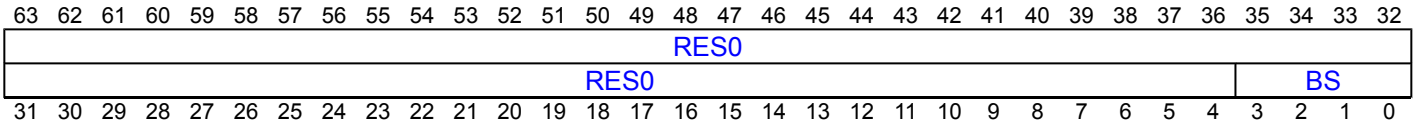
Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to GMID_EL1 are UNDEFINED.

Attributes

GMID_EL1 is a 64-bit register.

Field descriptions



Bits [63:4]

Reserved, RES0.

BS, bits [3:0]

Log_2 of the block size in words. The minimum supported size is 16B (value == 2) and the maximum is 256B (value == 6).

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GMID_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GMID_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b001	0b0000	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID5 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID5 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID5 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GMID_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID5 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID5 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GMID_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GMID_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GPCBW_EL3, Granule Protection Check Bypass Window Register (EL3)

The GPCBW_EL3 characteristics are:

Purpose

The control register for Granule Protection Check bypass window.

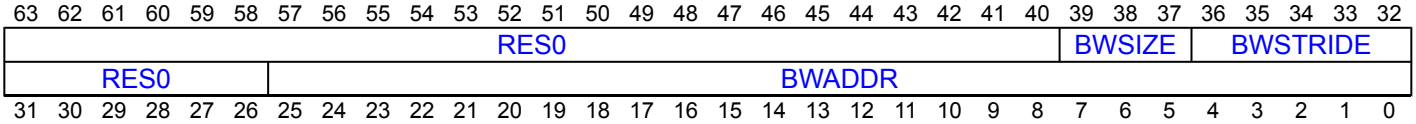
Configuration

This register is present only when FEAT_RME_GPC3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to GPCBW_EL3 are UNDEFINED.

Attributes

GPCBW_EL3 is a 64-bit register.

Field descriptions



Bits [63:40]

Reserved, RES0.

BWSIZE, bits [39:37]

GPC Bypass Window Size.

BWSIZE defines the size of the GPC bypass memory region.

BWSIZE	Meaning
0b000	30 bits, 1GB GPC bypass window.
0b001	31 bits, 2GB GPC bypass window.
0b010	32 bits, 4GB GPC bypass window.
0b100	34 bits, 16GB GPC bypass window.
0b110	36 bits, 64GB GPC bypass window.

All other values are reserved.

This field is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BWSTRIDE, bits [36:32]

GPC Bypass Window Stride.

BWSTRIDE allows creating multiple GPC bypass memory regions in the memory map across a specific stride.

BWSTRIDE	Meaning
0b00000	1TB stride.
0b00010	4TB stride.
0b00100	16TB stride.
0b00110	64TB stride.
0b00111	128TB stride.
0b01000	256TB stride.
0b01001	512TB stride.
0b01010	1PB stride.
0b10000	64PB (No stride).

All other values are reserved.

This field is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [31:26]

Reserved, RES0.

BWADDR, bits [25:0]

GPC Bypass window address.

This field represents bits [55:30] of the GPC bypass window base address.

The GPC bypass window is:

- Aligned in memory to the size of the window as specified by GPCBW_EL3.BWSIZE.
- Duplicated in PA space across a stride specified using GPCBW_EL3.BWSTRIDE.

This means that only bits [gpcbwu:gpcbwl] of a PA are compared against bits [gpcbwu:gpcbwl] of the window base address derived from BWADDR when checking if a PA falls within the range of a window, where:

- gpcbwl is derived from GPCBW_EL3.BWSIZE as follows:

BWSIZE	gpcbwl
0b000	30
0b001	31
0b010	32
0b100	34
0b110	36

- gpcbwu is derived from GPCBW_EL3.BWSTRIDE as follows:

BWSTRIDE	gpcbwu
0b00000	39
0b00010	41
0b00100	43
0b00110	45
0b00111	46
0b01000	47
0b01001	48
0b01010	49
0b10000	55

If the base address derived from BWADDR is not aligned to the size programmed in BWSIZE the configuration is invalid.

If this field is configured to a value that produces a GPC bypass window base address that is greater than or equal to the stride value configured by BWSTRIDE, the configuration is invalid.

An access to a PA falls within a GPC bypass window if the pseudocode function `PAWithinGPCBypassWindow()` returns TRUE.

This field is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GPCBW_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GPCBW_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_RME_GPC3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GPCBW_EL3;
```

```
MSR GPCBW_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_RME_GPC3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.GPCBW_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GPCBW_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GPCCR_EL3, Granule Protection Check Control Register (EL3)

The GPCCR_EL3 characteristics are:

Purpose

The control register for Granule Protection Checks.

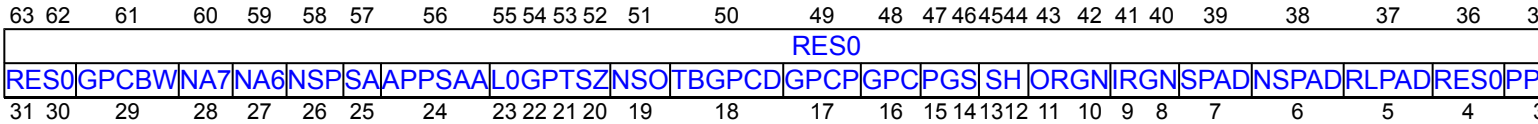
Configuration

This register is present only when FEAT_RME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to GPCCR_EL3 are UNDEFINED.

Attributes

GPCCR_EL3 is a 64-bit register.

Field descriptions



Bits [63:30]

Reserved, RES0.

GPCBW, bit [29]

When FEAT_RME_GPC3 is implemented:

GPC Bypass Window Enable.

This field governs the behavior of the GPC bypass windows.

GPCBW	Meaning
0b0	GPC bypass windows are disabled.
0b1	GPC bypass windows are enabled.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NA7, bit [28]

When FEAT_RME_GDI is implemented:

No access 7.

This field governs the behavior of the GPI encoding for NA7.

NA7	Meaning
0b0	GPI encoding value of 0b0111 is reserved.
0b1	GPI encoding value of 0b0111 is No access permitted from this PE.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NA6, bit [27]
When FEAT_RME_GDI is implemented:

No access 6.

This field governs the behavior of the GPI encoding for NA6.

NA6	Meaning
0b0	GPI encoding value of 0b0110 is reserved.
0b1	GPI encoding value of 0b0110 is No access permitted from this PE.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSP, bit [26]
When FEAT_RME_GDI is implemented:

Non-secure Protected.

This field governs the behavior of the GPI encoding for NSP.

NSP	Meaning
0b0	GPI encoding value of 0b0101 is reserved.
0b1	GPI encoding value of 0b0101 is No access permitted from this PE.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SA, bit [25]

When FEAT_RME_GDI is implemented:

System Agent.

This field governs the behavior of the GPI encoding for SA.

SA	Meaning
0b0	GPI encoding value of 0b0100 is reserved.
0b1	GPI encoding value of 0b0100 is No access permitted from this PE.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APPSAA, bit [24]

When FEAT_RME_GPC2 is implemented:

Above PPS All Access. This field governs the behavior of memory accesses to Secure, Realm and Root PA space, for physical addresses above the range configured by GPCCR_EL3.PPS.

APPSAA	Meaning
0b0	Accesses to addresses above the configured PPS must be to Non-secure PA space, otherwise they generate a GPF at level 0.
0b1	Accesses to addresses above the configured PPS, to any PA space, do not generate a GPF because of this control.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

L0GPTSZ, bits [23:20]

Level 0 GPT entry size.

This field advertises the number of least-significant address bits protected by each entry in the level 0 GPT.

L0GPTSZ	Meaning
0b0000	30-bits. Each entry covers 1GB of address space.
0b0100	34-bits. Each entry covers 16GB of address space.
0b0110	36-bits. Each entry covers 64GB of address space.
0b1001	39-bits. Each entry covers 512GB of address space.

All other values are reserved.

Access to this field is **RO**.

NSO, bit [19]

When FEAT_RME_GPC2 is implemented:

Non-secure Only. This field governs the behavior of the GPI encoding for NSO.

NSO	Meaning
0b0	GPI encoding value of 0b1101 is reserved.
0b1	GPI encoding value of 0b1101 is NSO.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBGPCD, bit [18]

When FEAT_TRBE_EXT is implemented:

Trace Buffer Granule Protection Check Disabled. Controls whether the Trace Buffer Unit accepts or rejects trace when Granule Protection Checks are disabled.

TBGPCD	Meaning
0b0	The Trace Buffer Unit rejects trace when GPCCR_EL3.GPC is 0.
0b1	The Trace Buffer Unit accepts trace when GPCCR_EL3.GPC is 0.

When the Trace Buffer Unit rejects trace, the trace might remain buffered by the trace unit until the Trace Buffer Unit is able to accept trace. When the Trace Buffer Unit accepts trace, the Trace Buffer Unit writes the trace to memory.

Note

Setting GPCCR_EL3.{TBGPCD, GPC} to {1, 0} means that the Trace Buffer Unit might write to memory without any Granule Protection Checks. The addresses that the Trace Buffer Unit writes to can be programmed by an external agent. The physical address spaces the Trace Buffer Unit can address are restricted by an IMPLEMENTATION DEFINED debug authentication interface.

Setting GPCCR_EL3.{TBGPCD, GPC} to {1, 1} means that GPCCR_EL3.{TBGPCD, GPC} will become {1, 0} on a Warm reset.

This field is ignored by the PE and treated as one when any of the following are true:

- GPCCR_EL3.GPC == 1.
- `ExternalRootInvasiveDebugEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

GPCP, bit [17]

Granule Protection Check Priority.

This control governs behavior of granule protection checks on fetches of stage 2 Table descriptors.

GPCP	Meaning
0b0	GPC faults are all reported with a priority that is consistent with the GPC being performed on any access to physical address space.
0b1	A GPC fault for the fetch of a Table descriptor for a stage 2 translation table walk might not be generated or reported. All other GPC faults are reported with a priority consistent with the GPC being performed on all accesses to physical address spaces.

This bit is permitted to be cached in a TLB.

An implementation is permitted to treat this field as RES0, with an Effective value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GPC, bit [16]

Granule Protection Check Enable.

GPC	Meaning
0b0	Granule protection checks are disabled. Accesses are not prevented by this mechanism.
0b1	All accesses to physical address spaces are subject to granule protection checks, except for fetches of GPT information and accesses governed by the GPCCR_EL3.GPCP control.

If any stage of translation is enabled, this bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

PGS, bits [15:14]

Physical Granule size.

PGS	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

All other values are reserved.

The value of this field is permitted to be cached in a TLB.

Granule sizes not supported for stage 1 and not supported for stage 2, as defined in ID_AA64MMFR0_EL1, are reserved. For example, if ID_AA64MMFR0_EL1.TGran16 == 0b0000 and ID_AA64MMFR0_EL1.TGran16_2 == 0b0001, then the PGS encoding 0b10 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [13:12]

GPT fetch Shareability attribute

SH	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

Fetches of GPT information are made with the Shareability attribute that is configured in this field.

If both ORGN and IRGN are configured with Non-cacheable attributes, it is invalid to configure this field to any value other than 0b10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN, bits [11:10]

GPT fetch Outer cacheability attribute.

ORGN	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

Fetches of GPT information are made with the Outer cacheability attributes configured in this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN, bits [9:8]

GPT fetch Inner cacheability attribute.

IRGN	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

Fetches of GPT information are made with the Inner cacheability attributes configured in this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SPAD, bit [7]

When FEAT_RME_GPC2 is implemented:

Secure PA space Disable. This field controls access to the Secure PA space.

SPAD	Meaning
0b0	This control has no effect on accesses.
0b1	When granule protection checks are enabled, access to the Secure Physical Address space generates a Granule Protection fault.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSPAD, bit [6]

When FEAT_RME_GPC2 is implemented:

Non-secure PA space Disable. This field controls access to the Non-secure PA space.

NSPAD	Meaning
0b0	This control has no effect on accesses.
0b1	When granule protection checks are enabled, access to the Non-secure Physical Address space generates a Granule Protection fault.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLPAD, bit [5]

When FEAT_RME_GPC2 is implemented:

Realm PA space Disable. This field controls access to the Realm PA space.

RLPAD	Meaning
0b0	This control has no effect on accesses.
0b1	When granule protection checks are enabled, access to the Realm Physical Address space generates a Granule Protection fault.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [4]

Reserved, RES0.

PPS3, bit [3]

When FEAT_RME_GPC3 is implemented:

This field extends GPCCR_EL3.PPS[2:0], creating a GPCCR_EL3.PPS[3:0] field.

For a description of the values derived by evaluating PPS, see GPCCR_EL3.PPS[2:0].

The value of this field is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PPS, bits [2:0]
When FEAT_RME_GPC3 is implemented:

Protected Physical Address Size.

The size of the memory region protected by GPTBR_EL3, in terms of the number of least-significant address bits.

This field is evaluated with PPS3, as {PPS3, PPS} to give PPS[3:0], interpreted as:

PPS[3:0]	Meaning
0b0000	32 bits, 4GB protected address space.
0b0001	36 bits, 64GB protected address space.
0b0010	40 bits, 1TB protected address space.
0b0011	42 bits, 4TB protected address space.
0b0100	44 bits, 16TB protected address space.
0b1000	46 bits, 64TB protected address space.
0b1001	47 bits, 128TB protected address space.
0b0101	48 bits, 256TB protected address space.
0b0110	52 bits, 4PB protected address space.
0b0111	56 bits, 64PB protected address space.

All other values are reserved.

Configuration of this field to a value exceeding the implemented physical address size is invalid.

The value of this field is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Protected Physical Address Size.

The size of the memory region protected by GPTBR_EL3, in terms of the number of least-significant address bits.

PPS	Meaning
0b000	32 bits, 4GB protected address space.
0b001	36 bits, 64GB protected address space.
0b010	40 bits, 1TB protected address space.
0b011	42 bits, 4TB protected address space.
0b100	44 bits, 16TB protected address space.
0b101	48 bits, 256TB protected address space.
0b110	52 bits, 4PB protected address space.

All other values are reserved.

Configuration of this field to a value exceeding the implemented physical address size is invalid.

The value of this field is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GPCCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, GPCCR_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0001	0b110

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GPCCR_EL3;
```

MSR GPCCR_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0001	0b110

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.GPCCR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GPCCR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GPTBR_EL3, Granule Protection Table Base Register

The GPTBR_EL3 characteristics are:

Purpose

The control register for Granule Protection Table base address.

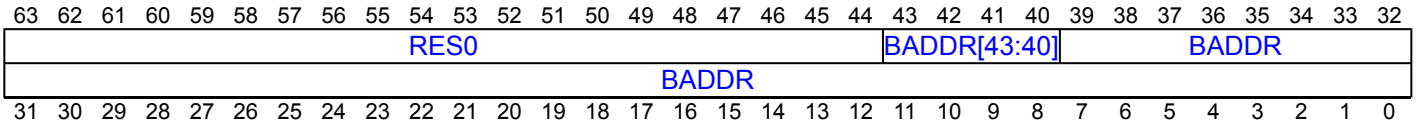
Configuration

This register is present only when FEAT_RME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to GPTBR_EL3 are UNDEFINED.

Attributes

GPTBR_EL3 is a 64-bit register.

Field descriptions



Bits [63:44]

Reserved, RES0.

BADDR[43:40], bits [43:40]

When FEAT_RME_GPC3 is implemented:

Extension to BADDR[39:0]. This field represents bit [55:52] of the level 0 GPT base address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BADDR, bits [39:0]

Base address for the level 0 GPT.

This field represents bits [51:12] of the level 0 GPT base address.

The level 0 GPT is aligned in memory to the greater of:

- The size of the level 0 GPT in bytes.
- 4KB.

Bits [x:0] of the base address are treated as zero, where:

- x = Max(pps - l0gptsz + 2, 11)
- pps is derived from GPCCR_EL3.PPS as follows:

GPCCR_EL3.PPS	pps
0b000	32
0b001	36
0b010	40
0b011	42
0b100	44
0b101	48
0b110	52

- l0gptsz is derived from GPCCR_EL3.L0GPTSZ as follows:

GPCCR_EL3.L0GPTSZ	l0gptsz
0b0000	30
0b0100	34
0b0110	36
0b1001	39

If x is greater than 11, then BADDR[x - 12:0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GPTBR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GPTBR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GPTBR_EL3;
```

```
MSR GPTBR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.GPTBR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GPTBR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HACDBSBR_EL2, Hardware Accelerator for Cleaning Dirty State Base Register

The HACDBSBR_EL2 characteristics are:

Purpose

Control register for HACDBS structure.

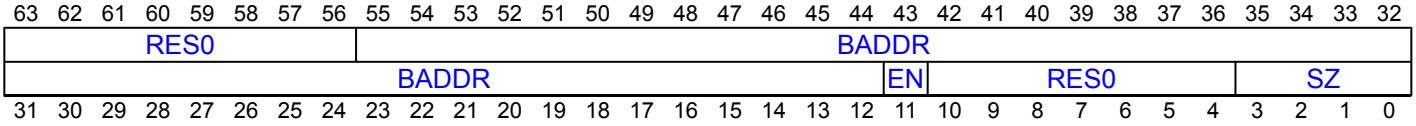
Configuration

This register is present only when FEAT_HACDBS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HACDBSBR_EL2 are UNDEFINED.

Attributes

HACDBSBR_EL2 is a 64-bit register.

Field descriptions



Bits [63:56]

Reserved, RES0.

BADDR, bits [55:12]

HACDBS base address, bits [55:12].

- Bits [55:12] of the base address are the value of this field.
- Bits [11:0] of the base address are 0.

Bits of this field above the implemented physical address size, indicated in ID_AA64MMFR0_EL1.PARange, are RES0.

Based on the value of the SZ field of this register, for encodings of the SZ field greater than 4KB, bits [(SZ+12-1):12] of this field are RES0 such that the base address of the HACDBS is aligned to its size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EN, bit [11]

Enable use of HACDBS.

EN	Meaning
0b0	Hardware accelerator for cleaning Dirty state is disabled.
0b1	Hardware accelerator for cleaning Dirty state is enabled.

If SCR_EL3.HACDBSEn is set to 0, then this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:4]

Reserved, RES0.

SZ, bits [3:0]

Size of the HACDBS.

SZ	Meaning
0b0000	4KB
0b0001	8KB
0b0010	16KB
0b0011	32KB
0b0100	64KB
0b0101	128KB
0b0110	256KB
0b0111	512KB
0b1000	1MB
0b1001	2MB

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HACDBSBR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HACDBSBR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0011	0b100

```
if !(IsFeatureImplemented(FEAT_HACDBS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x2F0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.HACDBSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HACDBSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HACDBSBR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HACDBSBR_EL2;
```

```
MSR HACDBSBR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0011	0b100

```
if !(IsFeatureImplemented(FEAT_HACDBS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x2F0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.HACDBSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HACDBSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HACDBSBR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HACDBSBR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HACDBSCONS_EL2, Hardware Accelerator for Cleaning Dirty State Consumer Register

The HACDBSCONS_EL2 characteristics are:

Purpose

Read index for HACDBS structure.

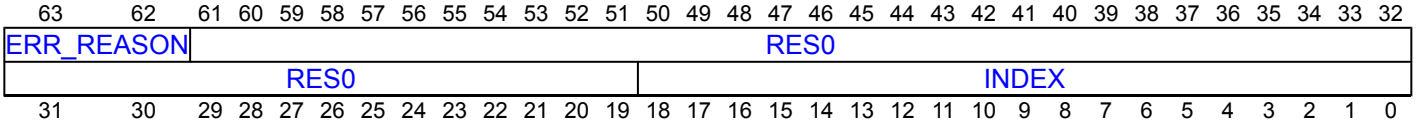
Configuration

This register is present only when FEAT_HACDBS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HACDBSCONS_EL2 are UNDEFINED.

Attributes

HACDBSCONS_EL2 is a 64-bit register.

Field descriptions



ERR_REASON, bits [63:62]

Reason for HACDBS error.

ERR_REASON	Meaning
0b00	The PE has not experienced an error while processing the HACDBS.
0b01	STRUCTF - A read of an entry from the HACDBS has experienced a fault.
0b10	IPAF - A stage 2 walk of an IPA from a HACDBS entry has experienced an MMU fault.
0b11	IPAHACF - An entry from the HACDBS experienced an error that is not an MMU fault.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [61:19]

Reserved, RES0.

INDEX, bits [18:0]

This field indicates the index of the HACDBS entry that will be read from next.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HACDBSCONS_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HACDBSCONS_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0011	0b101

```
if !(IsFeatureImplemented(FEAT_HACDBS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x308];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.HACDBSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HACDBSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HACDBSCONS_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HACDBSCONS_EL2;
```

```
MSR HACDBSCONS_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0011	0b101

```
if !(IsFeatureImplemented(FEAT_HACDBS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x308] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.HACDBSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HACDBSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HACDBSCONS_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HACDBSCONS_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HACR_EL2, Hypervisor Auxiliary Control Register

The HACR_EL2 characteristics are:

Purpose

Controls trapping to EL2 of IMPLEMENTATION DEFINED aspects of EL1 or EL0 operation.

Note

Arm recommends that the values in this register do not cause unnecessary traps to EL2 when the Effective value of HCR_EL2.{E2H, TGE} == {1, 1}.

Configuration

AArch64 System register HACR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HACR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to HACR_EL2 are UNDEFINED.

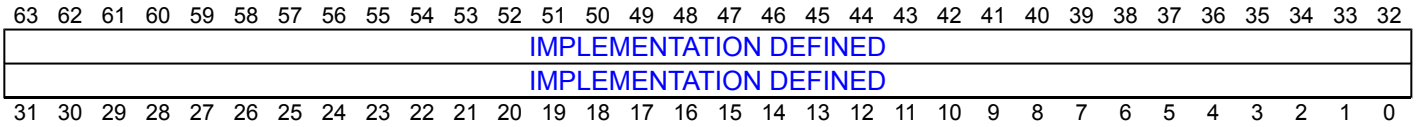
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

HACR_EL2 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HACR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HACR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = HACR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HACR_EL2;
```

```
MSR HACR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HACR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HACR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HAFGRTR_EL2, Hypervisor Activity Monitors Fine-Grained Read Trap Register

The HAFGRTR_EL2 characteristics are:

Purpose

Provides controls for traps of MRS reads of Activity Monitors System registers.

Configuration

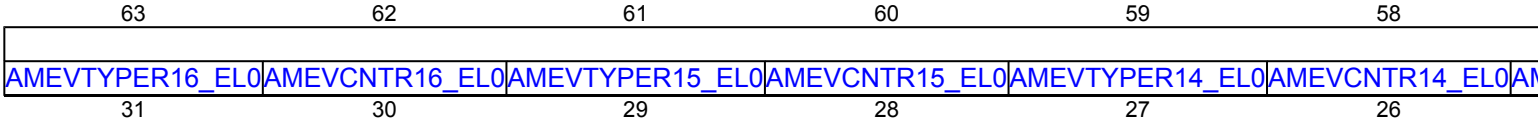
This register is present only when FEAT_AMUv1 is implemented, FEAT_FGT is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to HAFGRTR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HAFGRTR_EL2 is a 64-bit register.

Field descriptions



Bits [63:50]

Reserved, RES0.

AMEVTYPER1<x>_EL0, bit [19+2x], for x = 15 to 0

Trap MRS reads of AMEVTYPER1<x>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<x> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

AMEVTYPER1<x>_EL0	Meaning
0b0	MRS reads of AMEVTYPER1<x>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<x> at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception: • MRS reads of AMEVTYPER1<x>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of AMEVTYPER1<x> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When x >= UInt(AMCGCR_EL0.CG1NC), access to this field is **RES0**.
- When !IsG1ActivityMonitorImplemented(x), access to this field is **RES0**.

AMEVCNTR1<x>_EL0, bit [18+2x], for x = 15 to 0

Trap `MRS` reads of AMEVCNTR1<x>_EL0 at EL1 and EL0 using AArch64 and `MRC` reads of AMEVCNTR1<x> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

AMEVCNTR1<x>_EL0	Meaning
`0b0`	`MRS` reads of AMEVCNTR1<x>_EL0 at EL1 and EL0 using AArch64 and `MRC` reads of AMEVCNTR1<x> at EL0 using AArch32 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception: • `MRS` reads of AMEVCNTR1<x>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`. • `MRC` reads of AMEVCNTR1<x> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value `0x03`.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When x >= UInt(AMCGCR_EL0.CG1NC), access to this field is **RES0**.
- When !IsG1ActivityMonitorImplemented(x), access to this field is **RES0**.

AMCNTEN<x>, bit [17x], for x = 1 to 0

Trap `MRS` reads and `MRC` reads of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: `MRS` reads of AMCNTENCLR<x>_EL0 and AMCNTENSET<x>_EL0.
- At EL0 using AArch32 when EL1 is using AArch64: `MRC` reads of AMCNTENCLR<x> and AMCNTENSET<x>.

AMCNTEN<x>	Meaning
`0b0`	The operations listed above are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception: • `MRS` reads at EL1 and EL0 using AArch64 of AMCNTENCLR<x>_EL0 and AMCNTENSET<x>_EL0 are trapped to EL2 and reported with EC syndrome value `0x18`. • `MRC` reads at EL0 using AArch32 of AMCNTENCLR<x> and AMCNTENSET<x> are trapped to EL2 and reported with EC syndrome value `0x03`.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [16:5]

Reserved, RES0.

AMEVCNTR0<x>_EL0, bit [x+1], for x = 3 to 0

Trap MRS reads of AMEVCNTR0<x>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0<x> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

AMEVCNTR0<x>_EL0	Meaning
0b0	MRS reads of AMEVCNTR0<x>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0<x> at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception: • MRS reads of AMEVCNTR0<x>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of AMEVCNTR0<x> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

When x >= 4, access to this field is **RES0**.

Accessing HAFGRTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HAFGRTR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b110

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_FGT) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1E8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HAFGRTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HAFGRTR_EL2;
```

MSR HAFGRTR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b110

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_FGT) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1E8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HAFGRTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HAFGRTR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HCR_EL2, Hypervisor Configuration Register

The HCR_EL2 characteristics are:

Purpose

Provides configuration controls for virtualization, including defining whether various operations are trapped to EL2.

Configuration

AArch64 System register HCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HCR[31:0].

AArch64 System register HCR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HCR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to HCR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Unless otherwise stated, the bits in this register behave as if they are 0 for all purposes other than direct reads of the register if EL2 is not enabled in the current Security state.

Attributes

HCR_EL2 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43
TWEDEL				TWEDEn	TID5	DCT	ATA	TTLBOS	TTLBIS	EnSCXT	TOCU	AMVOFFEN	TICAB	TID4	GPF	FIEN	FWB	NV2	AT	NV1
RW	TRVM	HCD	TDZ	TGE	TVM	TTLB	TPU	TPCP	TSW	TACR	TIDCP	TSC	TID3	TID2	TID1	TID0	TWE	TWI	DC	BS
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11

TWEDEL, bits [63:60]

When FEAT_TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when HCR_EL2.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by HCR_EL2.TWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [59]

When FEAT_TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by HCR_EL2.TWE.

TWEDEn	Meaning
0b0	The delay for taking the trap is IMPLEMENTATION DEFINED.
0b1	The delay for taking the trap is at least the number of cycles defined in HCR_EL2.TWEDEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TID5, bit [58]

When FEAT_MTE2 is implemented:

Trap ID group 5. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

AArch64:

- GMID_EL1.

TID5	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL1 accesses to ID group 5 registers are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DCT, bit [57]

When FEAT_MTE2 is implemented:

Default Cacheability Tagging. When HCR_EL2.DC is in effect, controls whether EL1&0 stage 1 translations have the Tagged attribute.

DCT	Meaning
0b0	Stage 1 translations do not have the Tagged attribute.
0b1	Stage 1 translations have the Tagged attribute.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA, bit [56]

When FEAT_MTE2 is implemented:

Allocation Tag Access. Controls access to Allocation Tags, System registers for Memory tagging, and prevention of Tag checking, at EL1 and EL0

ATA	Meaning
0b0	Access to Allocation Tags is prevented at EL1 and EL0. Accesses at EL1 to GCR_EL1, RGSR_EL1, TFSR_EL1, or TFSRE0_EL1 that are not UNDEFINED are trapped to EL2. Accesses at EL1 using MRS or MSR with the register name TFSR_EL2 that are not UNDEFINED are trapped to EL2. Memory accesses at EL1 and EL0 are not subject to a Tag Check operation.
0b1	This control does not prevent access to Allocation Tags at EL1 and EL0. This control does not prevent Tag checking at EL1 and EL0.

If EL2 is not enabled in the current Security state, the Effective value of this field is 1.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTLBOS, bit [55]
When FEAT_EVT is implemented:

Trap TLB maintenance instructions that operate on the Outer Shareable domain. Traps execution of those TLB maintenance instructions at EL1 using AArch64 to EL2, when EL2 is enabled in the current Security state. The following instructions are trapped and reported with EC syndrome value 0x18:

- TLBI VMALLE1OS, TLBI VAE1OS, TLBI ASIDE1OS,TLBI VAAE1OS, TLBI VALE1OS, and TLBI VAALE1OS.
- If FEAT_TLBIRANGE is implemented, TLBI RVAE1OS, TLBI RVAAE1OS,TLBI RVALE1OS, and TLBI RVAALE1OS.
- If FEAT_XS is implemented then the *OSNXS variants are also trapped.

TTLBOS	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Execution of the specified instructions are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTLBIS, bit [54]
When FEAT_EVT is implemented:

Trap TLB maintenance instructions that operate on the Inner Shareable domain. Traps execution of those TLB maintenance instructions at EL1 to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64, then the following instructions are trapped to EL2 and reported with EC syndrome value 0x18:
  - TLBI VMALLE1IS, TLBI VAE1IS, TLBI ASIDE1IS, TLBI VAAE1IS, TLBI VALE1IS, TLBI VAALE1IS, TLBI RVAE1IS, TLBI RVAAE1IS, TLBI RVALE1IS, and TLBI RVAALE1IS.
- If EL1 is using AArch32, then the following instructions are trapped to EL2 and reported with EC syndrome value 0x03:
  - TLBIALLIS, TLBIMVAIS, TLBIASIDIS, TLBIMVAAIS, TLBIMVALIS, and TLBIMVAALIS.
- If FEAT_XS is implemented then the *ISNXS variants are also trapped.

TTLBIS	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Execution of the specified instructions are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSCXT, bit [53]
When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Enable Access to the SCXTNUM_EL1 and SCXTNUM_EL0 registers. The defined values are:

EnSCXT	Meaning
0b0	When EL2 is enabled in the current Security state, EL1 accesses to SCXTNUM_EL0 and SCXTNUM_EL1 are disabled, causing an exception to EL2, and the value of the registers to be treated as 0. When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1} and EL2 is enabled in the current Security state, EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL2, and the value of the register to be treated as 0.
0b1	This control does not cause accesses to SCXTNUM_EL0 or SCXTNUM_EL1 to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1} and the value of this field is 0, accesses at EL0 are not trapped by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TOCU, bit [52]
When FEAT_EVT is implemented:

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of those cache maintenance instructions at EL0 and EL1 to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL0 is using AArch64, the value of SCTLR_EL1.UCI is 1, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then the following instructions at EL0 are trapped to EL2 and reported with EC syndrome value 0x18:
  - IC IVAU, and DC CVAU.
- If EL1 is using AArch64, then the following instructions at EL1 are trapped to EL2 and reported with EC syndrome value 0x18:
  - IC IVAU, IC IALLU, and DC CVAU.
- If EL1 is using AArch32, then the following instructions are trapped at EL1 to EL2 and reported with EC syndrome value 0x03:
  - ICIMVAU, ICIALLU, and DCCMVAU.

Note

When SCTLR_EL1.UCI is 0, the trap on execution of instructions at EL0 is higher priority than this control.

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:

- IC IALLUIS and IC IALLU are always UNDEFINED at EL0 using AArch64.
- ICIMVAU, ICIALLU, ICIALLUIS, and DCCMVAU are always UNDEFINED at EL0 using AArch32.

TOCU	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	For each of the specified instructions, if the execution of the instruction can be trapped, accesses are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMVOFFEN, bit [51]
When FEAT_AMUv1p1 is implemented:

Activity Monitors Virtual Offsets Enable.

AMVOFFEN	Meaning
0b0	Virtualization of the Activity Monitors is disabled. Indirect reads of the virtual offset registers are zero.
0b1	Virtualization of the Activity Monitors is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TICAB, bit [50]
When FEAT_EVT is implemented:

Trap ICIALLUIS and IC IALLUIS cache maintenance instructions. Traps execution of those cache maintenance instructions at EL1 to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64, then the following instruction is trapped to EL2 and reported with EC syndrome value 0x18:
  - IC IALLUIS.
- If EL1 is using AArch32, then the following instruction is trapped to EL2 and reported with EC syndrome value 0x03:
  - ICIALLUIS.

TICAB	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	For each of the specified instructions, if the execution of the instruction can be trapped, EL1 access is trapped to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TID4, bit [49]
When FEAT_EVT is implemented:

Trap ID group 4. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

AArch64:

- EL1 reads of CCSIDR_EL1, CCSIDR2_EL1, CLIDR_EL1, and CSSELR_EL1.
- EL1 writes to CSSELR_EL1.

AArch32:

- EL1 reads of CCSIDR, CCSIDR2, CLIDR, and CSSELR.
- EL1 writes to CSSELR.

TID4	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL1 accesses to ID group 4 registers are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

GPF, bit [48]

When FEAT_RME is implemented:

Controls the reporting of Granule protection faults at EL0 and EL1.

GPF	Meaning
0b0	This control does not cause exceptions to be routed from EL0 and EL1 to EL2.
0b1	Instruction Abort exceptions and Data Abort exceptions due to GPFs from EL0 and EL1 are routed to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FIEN, bit [47]

When FEAT_RASv1p1 is implemented:

Fault Injection Enable. Unless this bit is set to 1, accesses to the ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1 registers from EL1 generate a Trap exception to EL2, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x18.

FIEN	Meaning
0b0	Accesses to the specified registers from EL1 are trapped to EL2, when EL2 is enabled in the current Security state.
0b1	This control does not cause any instructions to be trapped.

If EL2 is disabled in the current Security state, the Effective value of HCR_EL2.FIEN is 1.

If ERRIDR_EL1.NUM is zero, meaning no error records are implemented, or no error record accessible using System registers is owned by a node that implements the RAS Common Fault Injection Model Extension, then this bit might be RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FWB, bit [46]

When FEAT_S2FWB is implemented:

Forced Write-Back. Defines the combined cacheability attributes in a 2 stage translation regime.

FWB	Meaning
0b0	When this bit is 0, then the combination of stage 1 and stage 2 translations on memory type and cacheability attributes are as described in the Armv8.0 architecture. For more information, see 'Combining stage 1 and stage 2 memory type attributes'.
0b1	When this bit is 1, then the encoding of the stage 2 memory type and cacheability attributes in bits[5:2] of the stage 2 Page or Block descriptors are as described in 'Stage 2 memory type and Cacheability attributes when FEAT_S2FWB is enabled'.

In Secure state, this bit applies to both the Secure stage 2 translation and the Non-secure stage 2 translation.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NV2, bit [45]

When FEAT_NV2 is implemented:

Nested Virtualization. Changes the behaviors of HCR_EL2.{NV1, NV} to provide a mechanism for hardware to transform reads and writes from System registers into reads and writes from memory.

NV2	Meaning
0b0	This bit has no effect on the behavior of HCR_EL2.{NV1, NV}. The behavior of HCR_EL2.{NV1, NV} is as defined for FEAT_NV.
0b1	Redefines behavior of HCR_EL2{NV1, NV} to enable: • Transformation of read/writes to registers into read/writes to memory. • Redirection of EL2 registers to EL1 registers. Any exception taken from EL1 and taken to EL1 causes SPSR_EL1.M[3:2] to be set to 0b10 and not 0b01.

When the Effective value of HCR_EL2.NV is 0, the Effective value of this field is 0 and this field is treated as 0 for all purposes other than direct reads and writes of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AT, bit [44]

When FEAT_NV is implemented:

Address Translation. EL1 execution of the following address translation instructions is trapped to EL2, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x18:

- AT S1E0R, AT S1E0W, AT S1E1R, AT S1E1W, AT S1E1RP, and AT S1E1WP.
- If FEAT_ATS1A is implemented, AT S1E1A.

AT	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 execution of the specified instructions is trapped to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NV1, bit [43]
When FEAT_NV2 is implemented:

Nested Virtualization.

NV1	Meaning
0b0	If the Effective value of HCR_EL2.{NV2, NV} is {1, 1}, accesses executed from EL1 to implemented EL12, EL02, or EL2 registers are transformed to loads and stores. If the Effective value of HCR_EL2.{NV2, NV} is not {1, 1}, this control does not cause any instructions to be trapped.
0b1	If the Effective value of HCR_EL2.NV2 is 1, accesses executed from EL1 to implemented EL2 registers are transformed to loads and stores. If the Effective value of HCR_EL2.NV2 is 0, EL1 accesses to VBAR_EL1, ELR_EL1, SPSR_EL1, and, when FEAT_CSV2_2 or FEAT_CSV2_1p2 is implemented, SCXTNUM_EL1, are trapped to EL2, when EL2 is enabled in the current Security state, and are reported using EC syndrome value 0x18.

If the Effective value of HCR_EL2.NV2 is 1, the Effective value of HCR_EL2.NV1 defines which EL1 register accesses are transformed to loads and stores.

The trapping of EL1 registers caused by other control bits has priority over the transformation of these accesses.

If a register is specified that is not implemented by an implementation, then access to that register are UNDEFINED.

For the list of registers affected, see 'Enhanced support for nested virtualization'.

If the Effective value of HCR_EL2.{NV1, NV} is {0, 1}, any exception taken from EL1, and taken to EL1, causes the SPSR_EL1.M[3:2] to be set to 0b10, and not 0b01.

If the Effective value of HCR_EL2.{NV1, NV} is {1, 1}, then:

- The EL1 translation table Block and Page descriptors:
  - Bit[54] holds the PXN instead of the UXN.
  - The Effective value of UXN is 0.
  - Bit[53] is RES0.
  - Bit[6] is treated as 0 regardless of the actual value.
- If Hierarchical Permissions are enabled, the EL1 translation table Table descriptors are as follows:
  - Bit[61] is treated as 0 regardless of the actual value.
  - Bit[60] holds the PXNTable instead of the UXNTable.
  - Bit[59] is RES0.
- When executing at EL1, the PSTATE.PAN bit is treated as zero for all purposes except reading the value of the bit.
- When executing at EL1, the LDTR* instructions are treated as the equivalent LDR* instructions, and the STTR* instructions are treated as the equivalent STR* instructions.

If the Effective value of HCR_EL2.{NV1, NV} are {1, 0}, then the behavior is a CONSTRAINED UNPREDICTABLE choice of:

- Behaving as if the Effective value of HCR_EL2.{NV1, NV} is {1, 1} for all purposes other than reading back the value of the HCR_EL2.NV bit.
- Behaving as if the Effective value of HCR_EL2.{NV1, NV} is {0, 0} for all purposes other than reading back the value of the HCR_EL2.NV1 bit.
- Behaving with regard to the Effective value of HCR_EL2.{NV1, NV} behavior as defined in the rest of this description.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_NV is implemented:

Nested Virtualization. EL1 accesses to certain registers are trapped to EL2, when EL2 is enabled in the current Security state.

NV1	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to VBAR_EL1, ELR_EL1, SPSR_EL1, and, when FEAT_CSV2_2 or FEAT_CSV2_1p2 is implemented, SCXTNUM_EL1, are trapped to EL2, when EL2 is enabled in the current Security state, and are reported using EC syndrome value 0x18.

If the Effective value of HCR_EL2.{NV1, NV} is {0, 1}, then the following effects also apply:

- Any exception taken from EL1, and taken to EL1, causes the SPSR_EL1.M[3:2] to be set to 0b10, and not 0b01.

If the Effective value of HCR_EL2.{NV1, NV} is {1, 1}, then the following effects also apply:

- The EL1 translation table Block and Page descriptors:
  - Bit[54] holds the PXN instead of the UXN.
  - The Effective value of UXN is 0.
  - Bit[53] is RES0.
  - Bit[6] is treated as 0 regardless of the actual value.
- If Hierarchical Permissions are enabled, the EL1 translation table Table descriptors are as follows:
  - Bit[61] is treated as 0 regardless of the actual value.
  - Bit[60] holds the PXNTable instead of the UXNTable.
  - Bit[59] is RES0.
- When executing at EL1, the PSTATE.PAN bit is treated as zero for all purposes except reading the value of the bit.
- When executing at EL1, the LDTR* instructions are treated as the equivalent LDR* instructions, and the STTR* instructions are treated as the equivalent STR* instructions.

If the Effective value of HCR_EL2.{NV1, NV} is {1, 0}, then the behavior is a CONSTRAINED UNPREDICTABLE choice of:

- Behaving as if the Effective value of HCR_EL2.{NV1, NV} is {1, 1} for all purposes other than reading back the value of the HCR_EL2.NV bit.
- Behaving as if the Effective value of HCR_EL2.{NV1, NV} is {0, 0} for all purposes other than reading back the value of the HCR_EL2.NV1 bit.
- Behaving with regard to the Effective value of HCR_EL2.{NV1, NV} behavior as defined in the rest of this description.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NV, bit [42]
When FEAT_NV2 is implemented:

Nested Virtualization.

When the Effective value of HCR_EL2.NV2 is 1, redefines register accesses so that:

- Instructions accessing the Special purpose registers SPSR_EL2 and ELR_EL2 instead access SPSR_EL1 and ELR_EL1 respectively.
- Instructions accessing the System registers ESR_EL2 and FAR_EL2 instead access ESR_EL1 and FAR_EL1.

When the Effective value of HCR_EL2.NV2 is 0, traps functionality that is permitted at EL2 and would be UNDEFINED at EL1 if this field was 0, when EL2 is enabled in the current Security state. This applies to the following operations:

- EL1 accesses to Special-purpose registers that are not UNDEFINED at EL2.
- EL1 accesses to System registers that are not UNDEFINED at EL2.
- Execution of EL1 or EL2 translation regime address translation and TLB maintenance instructions for EL2 and above.

NV	Meaning
0b0	When this bit is set to 0, then the PE behaves as if the Effective value of HCR_EL2.NV2 is 0 for all purposes other than reading this register. This control does not cause any instructions to be trapped. When the Effective value of HCR_EL2.NV2 is 1, no FEAT_NV2 functionality is implemented.
0b1	When the Effective value of HCR_EL2.NV2 is 0, EL1 accesses to the specified registers or the execution of the specified instructions are trapped to EL2, when EL2 is enabled in the current Security state. EL1 read accesses to the CurrentEL register return a value of 0x2. When the Effective value of HCR_EL2.NV2 is 1, this control redefines EL1 register accesses so that instructions accessing SPSR_EL2, ELR_EL2, ESR_EL2, and FAR_EL2 instead access SPSR_EL1, ELR_EL1, ESR_EL1, and FAR_EL1 respectively.

When the Effective value of HCR_EL2.NV2 is 0, then:

- The System or Special-purpose registers for which accesses are trapped and reported using EC syndrome value 0x18 are as follows:
  - Registers accessed using MRS or MSR with a name ending in _EL2, except the following:
    - SP_EL2.
    - If FEAT_MEC is implemented, MECID_A0_EL2, MECID_A1_EL2, MECID_P0_EL2, MECID_P1_EL2, MECIDR_EL2, VMECID_A_EL2, and VMECID_P_EL2.
  - Registers accessed using MRS or MSR with a name ending in _EL12.
  - Registers accessed using MRS or MSR with a name ending in _EL02.
  - Special-purpose registers SPSR_irq, SPSR_abt, SPSR_und, and SPSR_fiq, accessed using MRS or MSR.
  - Special-purpose register SP_EL1 accessed using the dedicated MRS or MSR instruction.
- The instructions for which the execution is trapped and reported using EC syndrome value 0x18 are as follows:
  - EL2 translation regime Address Translation instructions and TLB maintenance instructions.
  - EL1 translation regime Address Translation instructions and TLB maintenance instructions that are accessible only from EL2 and EL3.
- The instructions for which the execution is trapped as follows:
  - SMC in an implementation that does not include EL3 and when HCR_EL2.TSC is 1. HCR_EL2.TSC bit is not RES0 in this case. This is reported using EC syndrome value 0x17.
  - The ERET, ERETAA, and ERETAB instructions, reported using EC syndrome value 0x1A.

Note

The priority of this trap is higher than the priority of the HCR_EL2.API trap. If both of these bits are set so that EL1 execution of an ERETAA or ERETAB instruction is trapped to EL2, then the syndrome reported is 0x1A.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_NV is implemented:

Nested Virtualization. Traps functionality that is permitted at EL2 and would be UNDEFINED at EL1 if this field was 0, when EL2 is enabled in the current Security state. This applies to the following operations:

- EL1 accesses to Special-purpose registers that are not UNDEFINED at EL2.
- EL1 accesses to System registers that are not UNDEFINED at EL2.
- Execution of EL1 or EL2 translation regime address translation and TLB maintenance instructions for EL2 and above.

NV	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the specified registers or the execution of the specified instructions are trapped to EL2, when EL2 is enabled in the current Security state. EL1 read accesses to the CurrentEL register return a value of 0x2.

The System or Special-purpose registers for which accesses are trapped and reported using EC syndrome value 0x18 are as follows:

- Registers accessed using MRS or MSR with a name ending in _EL2, except the following:
  - SP_EL2.

        - If FEAT_MEC is implemented, MECID_A0_EL2, MECID_A1_EL2, MECID_P0_EL2, MECID_P1_EL2, MECIDR_EL2, VMECID_A_EL2, and VMECID_P_EL2.
    - Registers accessed using MRS or MSR with a name ending in _EL12.
    - Registers accessed using MRS or MSR with a name ending in _EL02.
    - Special-purpose registers SPSR_irq, SPSR_abt, SPSR_und, and SPSR_fiq, accessed using MRS or MSR.
    - Special-purpose register SP_EL1 accessed using the dedicated MRS or MSR instruction.

The instructions for which the execution is trapped and reported using EC syndrome value `0x18` are as follows:

- EL2 translation regime Address Translation instructions and TLB maintenance instructions.
- EL1 translation regime Address Translation instructions and TLB maintenance instructions that are accessible only from EL2 and EL3.

The execution of the ERET, ERETAA, and ERETAB instructions are trapped and reported using EC syndrome value `0x1A`.

Note

The priority of this trap is higher than the priority of the HCR_EL2.API trap. If both of these bits are set so that EL1 execution of an ERETAA or ERETAB instruction is trapped to EL2, then the syndrome reported is `0x1A`.

The execution of the SMC instructions in an implementation that does not include EL3 and when HCR_EL2.TSC is 1 are trapped and reported using EC syndrome value `0x17`. HCR_EL2.TSC bit is not RES0 in this case.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

API, bit [41]
When FEAT_PAuth is implemented:

Controls the use of instructions related to Pointer Authentication:

- PACGA.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB.
- PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB.
- RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ.
- ERETAA, ERETAB, LDRAA, and LDRAB.
- When FEAT_PAuth_LR is implemented, AUTIASPPC, AUTIASPPCR, AUTIA171615, AUTIBSPPC, AUTIBSPPCR, AUTIB171615, PACIASPPC, PACNBIASPPC, PACIA171615, PACIBSPPC, PACNBIBSPPC, PACIB171615, RETAASPPC, RETAASPPCR, RETABSPPC, RETABSPPCR.

This field is ignored if the instruction is disabled as a result of the SCTLR_ELx.{EnIB, EnIA, EnDA, EnDB} fields.

API	Meaning
0b0	The instructions related to Pointer Authentication are trapped to EL2 and reported using EC syndrome value 0x09, when EL2 is enabled in the current Security state and the instructions are enabled for the EL1&0 translation regime, from: • When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, EL0. • EL1. If the Effective value of HCR_EL2.NV is 1, the HCR_EL2.NV trap takes precedence over the HCR_EL2.API trap for the ERETAA and ERETAB instructions. If EL2 is implemented and enabled in the current Security state and the Effective value of HFGITR_EL2.ERET is 1, execution at EL1 using AArch64 of ERETAA or ERETAB instructions is reported with EC syndrome value 0x1A with its associated ISS field, as the fine-grained trap has higher priority than the trap enabled by HCR_EL2.API == 0.
0b1	This control does not cause any instructions to be trapped.

If FEAT_PAuth is implemented but EL2 is not implemented or is disabled in the current Security state, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APK, bit [40]

When FEAT_PAuth is implemented:

Trap registers holding "key" values for Pointer Authentication. Traps accesses to the following registers from EL1 to EL2, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x18:

- APIAKeyLo_EL1, APIAKeyHi_EL1, APIBKeyLo_EL1, APIBKeyHi_EL1, APDAKeyLo_EL1, APDAKeyHi_EL1, APDBKeyLo_EL1, APDBKeyHi_EL1, APGAKeyLo_EL1, and APGAKeyHi_EL1.

APK	Meaning
0b0	Access to the registers holding "key" values for pointer authentication from EL1 are trapped to EL2, when EL2 is enabled in the current Security state.
0b1	This control does not cause any instructions to be trapped.

Note

If FEAT_PAuth is implemented but EL2 is not implemented or is disabled in the current Security state, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME, bit [39]

When FEAT_TME is implemented:

Enables access to the TSTART, TCOMMIT, TTEST, and TCANCEL instructions at EL0 and EL1.

TME	Meaning
0b0	EL0 and EL1 accesses to TSTART, TCOMMIT, TTEST, and TCANCEL instructions are UNDEFINED.
0b1	This control does not cause any instruction to be UNDEFINED.

If EL2 is not implemented or is disabled in the current Security state, the Effective value of this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [38]

Note

This bit was previously the MIOCNCE control.

Reserved, RES0.

TEA, bit [37]
When FEAT_RAS is implemented:

Route synchronous External abort exceptions to EL2.

TEA	Meaning
0b0	Synchronous External abort exceptions are unaffected by this mechanism. That is, synchronous External abort exceptions are not taken to EL2 unless routed to EL2 by another control.
0b1	When executing at Exception levels below EL2, and EL2 is enabled in the current Security state, synchronous External abort exceptions are taken to EL2, unless they are routed to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TERR, bit [36]
When FEAT_RAS is implemented:

Trap accesses of Error Record registers. Enables a trap to EL2 on accesses of Error Record registers.

TERR	Meaning
0b0	Accesses of the specified Error Record registers are not trapped by this mechanism.
0b1	Accesses of the specified Error Record registers at EL1 are trapped to EL2, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to ERRSELR_EL1, ERXADDR_EL1, ERXCTLR_EL1, ERXMISC0_EL1, ERXMISC1_EL1, and ERXSTATUS_EL1.
- MRS accesses to ERRIDR_EL1 and ERXFR_EL1.
- If FEAT_RASv1p1 is implemented, MRS and MSR accesses to ERXMISC2_EL1 and ERXMISC3_EL1.
- If FEAT_RASv2 is implemented, MRS accesses to ERXGSR_EL1.

In AArch32 state, the instructions affected by this control are:

- MRC and MCR accesses to ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS.
- MRC accesses to ERRIDR, ERXFR, and ERXFR2.
- If FEAT_RASv1p1 is implemented, MRC and MCR accesses to ERXMISC4, ERXMISC5, ERXMISC6, and ERXMISC7.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL2.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Trapped AArch32 instructions are reported using EC syndrome value 0x03.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLOR, bit [35]
When FEAT_LOR is implemented:

Trap LOR registers. Traps Non-secure and Realm EL1 accesses to LORSA_EL1, LOREA_EL1, LORN_EL1, LORC_EL1, and LORID_EL1 registers to EL2.

TLOR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure and Realm EL1 accesses to the LOR registers are trapped to EL2.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E2H, bit [34]
When FEAT_VHE is implemented:

EL2 Host. Enables a configuration where a Host Operating System is running in EL2, and the Host Operating System's applications are running in EL0.

E2H	Meaning
0b0	The facilities to support a Host Operating System at EL2 are disabled.
0b1	The facilities to support a Host Operating System at EL2 are enabled.

For information on the behavior of this bit see 'Behavior of HCR_EL2.E2H'.

When FEAT_E2H0 is not implemented, this field is RES1 and behaves as if it is 1 for all purposes other than a direct read.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ID, bit [33]

Stage 2 Instruction access cacheability disable. For the EL1&0 translation regime, when EL2 is enabled in the current Security state and HCR_EL2.VM==1, this control forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.

ID	Meaning
0b0	This control has no effect on stage 2 of the EL1&0 translation regime.
0b1	Forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.

This bit has no effect on the EL2, EL2&0, or EL3 translation regimes.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CD, bit [32]

Stage 2 Data access cacheability disable. For the EL1&0 translation regime, when EL2 is enabled in the current Security state and HCR_EL2.VM==1, this control forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable.

CD	Meaning
0b0	This control has no effect on stage 2 of the EL1&0 translation regime for data accesses and translation table walks.
0b1	Forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable.

This bit has no effect on the EL2, EL2&0, or EL3 translation regimes.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RW, bit [31]

When FEAT_AA32EL1 is implemented:

Execution state control for lower Exception levels:

RW	Meaning
0b0	Lower levels are all AArch32.
0b1	The Execution state for EL1 is AArch64. The Execution state for EL0 is determined by the current value of PSTATE.nRW when executing at EL0.

In an implementation that includes EL3, when EL2 is not enabled in Secure state, the PE behaves as if this bit has the same value as the SCR_EL3.RW bit for all purposes other than a direct read or write access of HCR_EL2.

The RW bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAO/WI.

TRVM, bit [30]

Trap Reads of Virtual Memory controls. Traps reads of the virtual memory control registers to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64, EL1 accesses to the following registers are trapped to EL2 and reported using EC syndrome value `0x18` for `MRS` and `0x14` for `MRRS`:
  - SCTLR_EL1, TTBR0_EL1, TTBR1_EL1, TCR_EL1, ESR_EL1, FAR_EL1, AFSR0_EL1, AFSR1_EL1, MAIR_EL1, AMAIR_EL1, and CONTEXTIDR_EL1.
  - If FEAT_AIE is implemented, MAIR2_EL1 and AMAIR2_EL1.
  - If FEAT_S1PIE is implemented, PIRE0_EL1 and PIR_EL1.
  - If FEAT_S1POE is implemented, POR_EL0 and POR_EL1.
  - If FEAT_S2POE is implemented, S2POR_EL1.
  - If FEAT_TCR2 is implemented, TCR2_EL1.
  - If FEAT_SCTLR2 is implemented, SCTLR2_EL1.
- If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and EL0 is using AArch64, EL0 accesses to the following registers are trapped to EL2 and reported using EC syndrome value `0x18` for `MRS`:
  - If FEAT_S1POE is implemented, POR_EL0.
- If EL1 is using AArch32, EL1 accesses using MRC to the following registers are trapped to EL2 and reported using EC syndrome value `0x03`, accesses using MRRC are trapped to EL2 and reported using EC syndrome value `0x04`:
  - SCTLR, TTBR0, TTBR1, TTBCR, TTBCR2, DACR, DFSR, IFSR, DFAR, IFAR, ADFSR, AIFSR, PRRR, NMRR, MAIR0, MAIR1, AMAIR0, AMAIR1, and CONTEXTIDR.

TRVM	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Read accesses to the specified Virtual Memory control registers are trapped to EL2, when EL2 is enabled in the current Security state.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

Note

EL2 provides a second stage of address translation, that a hypervisor can use to remap the address map defined by a Guest OS. In addition, a hypervisor can trap attempts by a Guest OS to write to the registers that control the memory system. A hypervisor might use this trap as part of its virtualization of memory management.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HCD, bit [29]

When EL3 is not implemented:

HVC instruction disable. Disables EL1 and EL2 execution of HVC instructions, from both Execution states, when EL2 is enabled in the current Security state, reported using EC syndrome value `0x00`.

HCD	Meaning
`0b0`	HVC instruction execution is enabled at EL2 and EL1.
`0b1`	HVC instructions are UNDEFINED at EL2 and EL1. Any resulting exception is taken to the Exception level at which the HVC instruction is executed.

Note

HVC instructions are always UNDEFINED at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TDZ, bit [28]

Trap DC ZVA instructions. Traps EL0 and EL1 execution of DC ZVA instructions to EL2, when EL2 is enabled in the current Security state, from AArch64 state only, reported using EC syndrome value `0x18`.

If FEAT_MTE is implemented, this trap also applies to DC GVA and DC GZVA.

TDZ	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	In AArch64 state, any attempt to execute an instruction this trap applies to at EL1, or at EL0 when the instruction is not UNDEFINED at EL0, is trapped to EL2 when EL2 is enabled in the current Security state. Reading the DCZID_EL0 returns a value that indicates that the instructions this trap applies to are not supported.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TGE, bit [27]

Trap General Exceptions, from EL0.

TGE	Meaning
`0b0`	This control has no effect on execution at EL0.
`0b1`	When EL2 is not enabled in the current Security state, this control has no effect on execution at EL0. When EL2 is enabled in the current Security state, in all cases: • All exceptions that would be routed to EL1 are routed to EL2. • If EL1 is using AArch64, the SCTLR_EL1.M field is treated as being 0 for all purposes other than returning the result of a direct read of SCTLR_EL1. • If EL1 is using AArch32, the SCTLR.M field is treated as being 0 for all purposes other than returning the result of a direct read of SCTLR. • All virtual interrupts and virtual exceptions are disabled. • Any IMPLEMENTATION DEFINED mechanisms for signaling virtual interrupts are disabled. • An exception return to EL1 is treated as an illegal exception return. • The MDCR_EL2.{TDRA, TDOSA, TDA, TDE} fields are treated as being 1 for all purposes other than returning the result of a direct read of MDCR_EL2. In addition, when EL2 is enabled in the current Security state, if: • The Effective value of HCR_EL2.E2H is not 1, the Effective values of the HCR_EL2.{FMO, IMO, AMO} fields are 1. • The Effective value of HCR_EL2.E2H is 1, the Effective values of the HCR_EL2.{FMO, IMO, AMO} fields are 0. For further information on the behavior of this bit when the Effective value of E2H is 1, see 'Behavior of HCR_EL2.E2H'.

HCR_EL2.TGE must not be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TVM, bit [26]

Trap Virtual Memory controls. Traps writes to the virtual memory control registers to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64, the following registers are trapped to EL2 and reported using EC syndrome value `0x18` for `MSR` and `0x14` for `MSRR`:
  - SCTLR_EL1, TTBR0_EL1, TTBR1_EL1, TCR_EL1, ESR_EL1, FAR_EL1, AFSR0_EL1, AFSR1_EL1, MAIR_EL1, AMAIR_EL1, and CONTEXTIDR_EL1.
  - If FEAT_AIE is implemented, MAIR2_EL1 and AMAIR2_EL1.
  - If FEAT_S1PIE is implemented, PIRE0_EL1 and PIR_EL1.
  - If FEAT_S1POE is implemented, POR_EL0 and POR_EL1.
  - If FEAT_S2POE is implemented, S2POR_EL1.
  - If FEAT_TCR2 is implemented, TCR2_EL1.
  - If FEAT_SCTLR2 is implemented, SCTLR2_EL1.
- If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and EL0 is using AArch64, EL0 accesses to the following registers are trapped to EL2 and reported using EC syndrome value `0x18` for `MSR`:
  - If FEAT_S1POE is implemented, POR_EL0.
- If EL1 is using AArch32, EL1 accesses using MCR to the following registers are trapped to EL2 and reported using EC syndrome value `0x03`, accesses using MCRR are trapped to EL2 and reported using EC syndrome value `0x04`:
  - SCTLR, TTBR0, TTBR1, TTBCR, TTBCR2, DACR, DFSR, IFSR, DFAR, IFAR, ADFSR, AIFSR, PRRR, NMRR, MAIR0, MAIR1, AMAIR0, AMAIR1, and CONTEXTIDR.

TVM	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Write accesses to the specified Virtual Memory control registers are trapped to EL2, when EL2 is enabled in the current Security state.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TTLB, bit [25]

Trap TLB maintenance instructions. Traps execution of TLB maintenance instructions at EL1 to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64, then the following instructions are trapped to EL2 and reported using EC syndrome value `0x18`:
  - TLBI VMALLE1, TLBI VAE1, TLBI ASIDE1, TLBI VAAE1, TLBI VALE1, and TLBI VAALE1.
  - TLBI VMALLE1IS, TLBI VAE1IS, TLBI ASIDE1IS, TLBI VAAE1IS, TLBI VALE1IS, and TLBI VAALE1IS.
  - If FEAT_TLBIOS is implemented, TLBI VMALLE1OS, TLBI VAE1OS, TLBI ASIDE1OS, TLBI VAAE1OS, TLBI VALE1OS, and TLBI VAALE1OS.
  - If FEAT_TLBIRANGE is implemented, TLBI RVAE1, TLBI RVAAE1, TLBI RVALE1, TLBI RVAALE1, TLBI RVAE1IS, TLBI RVAAE1IS, TLBI RVALE1IS, and TLBI RVAALE1IS.
  - If FEAT_TLBIOS and FEAT_TLBIRANGE are implemented, TLBI RVAE1OS, TLBI RVAAE1OS, TLBI RVALE1OS, and TLBI RVAALE1OS.
- If EL1 is using AArch32, then the following instructions are trapped to EL2 and reported using EC syndrome value `0x03`:
  - TLBIALLIS, TLBIMVAIS, TLBIASIDIS, TLBIMVAAIS, TLBIMVALIS, and TLBIMVAALIS.
  - TLBIALL, TLBIMVA, TLBIASID, TLBIMVAA, TLBIMVAL, and TLBIMVAAL
  - ITLBIALL, ITLBIMVA, and ITLBIASID.
  - DTLBIALL, DTLBIMVA, and DTLBIASID.

TTLB	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL1 execution of the specified TLB maintenance instructions are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

Note

The TLB maintenance instructions are UNDEFINED at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TPU, bit [24]

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of those cache maintenance instructions at EL0 and EL1 to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL0 is using AArch64 and the value of SCTLR_EL1.UCI is 1, then the following instructions at EL0 are trapped to EL2 and reported with EC syndrome value `0x18`:
  - IC IVAU and DC CVAU.
- If EL1 is using AArch64, then the following instructions at EL1 are trapped to EL2 and reported with EC syndrome value `0x18`:
  - IC IVAU, IC IALLU, IC IALLUIS, and DC CVAU.
- If EL1 is using AArch32, then the following instructions are trapped to EL2 and reported with EC syndrome value `0x03`:
  - ICIMVAU, ICIALLU, ICIALLUIS, and DCCMVAU.

Note

When SCTLR_EL1.UCI is 0, the trap on execution of instructions at EL0 is higher priority than this control.

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:

- IC IALLUIS and IC IALLU are always UNDEFINED at EL0 using AArch64.
- ICIMVAU, ICIALLU, ICIALLUIS, and DCCMVAU are always UNDEFINED at EL0 using AArch32.

TPU	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	For each of the specified instructions, if the execution of the instruction can be trapped, access is trapped to EL2, when EL2 is enabled in the current Security state.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TPCP, bit [23]

Trap data or unified cache maintenance instructions that operate to the Point of Coherency, Persistence, or Physical Storage. When EL2 is enabled in the current Security state, traps execution of cache maintenance instructions at EL0 and EL1 to EL2 as follows:

- If EL0 is using AArch64 and the value of SCTLR_EL1.UCI is 1, then the following instructions at EL0 are trapped to EL2 and reported using EC syndrome value `0x18`:
  - DC CIVAC and DC CVAC.
  - If FEAT_MTE is implemented, DC CIGVAC, DC CIGDVAC, DC CGVAC, and DC CGDVAC.
  - If FEAT_DPB is implemented, DC CVAP.
  - If FEAT_DPB and FEAT_MTE are implemented, DC CGVAP and DC CGDVAP.
  - If FEAT_DPB2 is implemented, DC CVADP.
  - If FEAT_DPB2 and FEAT_MTE are implemented, DC CGVADP and DC CGDVADP.
- If FEAT_OCCMO is implemented, DC CIVAOC, DC CIGDVAOC, DC CVAOC and DC CGDVAOC.
- If EL1 is using AArch64, then the following instructions at EL1 are trapped to EL2 and reported using EC syndrome value `0x18`:
  - DC IVAC, DC CIVAC and DC CVAC.
  - If FEAT_MTE is implemented, DC CIGVAC, DC CIGDVAC, DC IGVAC, DC IGDVAC, DC CGVAC, and DC CGDVAC.
  - If FEAT_DPB is implemented, DC CVAP.
  - If FEAT_DPB and FEAT_MTE are implemented, DC CGVAP and DC CGDVAP.
  - If FEAT_DPB2 is implemented, DC CVADP.
  - If FEAT_DPB2 and FEAT_MTE are implemented, DC CGVADP and DC CGDVADP.
  - If FEAT_PoPS is implemented, DC CIVAPS.
  - If FEAT_PoPS and FEAT_MTE2 are implemented, DC CIGDVAPS.

- If FEAT_OCCMO is implemented, DC CIVAOC, DC CIGDVAOC, DC CVAOC and DC CGDVAOC.
- If EL1 is using AArch32, then the following instructions at EL1 are trapped to EL2 and reported using EC syndrome value 0x03:
  - DCIMVAC, DCCIMVAC, and DCCMVAC.

Note

This field was previously named TPC.

When SCTLR_EL1.UCI is 0, the trap on execution of instructions at EL0 is higher priority than this control.

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:

- AArch64 instructions which invalidate by VA to the Point of Coherency or Physical Storage are always UNDEFINED at EL0 using AArch64.
- DCIMVAC, DCCIMVAC, and DCCMVAC are always UNDEFINED at EL0 using AArch32.

TPCP	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	For each of the specified instructions, if the execution of the instruction can be trapped, it is trapped to EL2, when EL2 is enabled in the current Security state.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TSW, bit [22]

Trap data or unified cache maintenance instructions that operate by Set/Way. Traps execution of those cache maintenance instructions at EL1 to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64, then the following instructions are trapped to EL2 and reported with EC syndrome value 0x18:
  - DC ISW, DC CSW, and DC CISW.
  - If FEAT_MTE2 is implemented, DC IGSW, DC IGDSW, DC CGSW, DC CGDW, DC CIGSW, and DC CIGDSW.
- If EL1 is using AArch32, then the following instructions are trapped to EL2 and reported with EC syndrome value 0x03:
  - DCISW, DCCSW, and DCCISW.

Note

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.

TSW	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TACR, bit [21]

Trap Auxiliary Control Registers. Traps EL1 accesses to the Auxiliary Control Registers to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64, accesses to ACTLR_EL1 to EL2, are trapped to EL2 and reported using EC syndrome value 0x18.
- If EL1 is using AArch32, accesses to ACTLR and, if implemented, ACTLR2 are trapped to EL2 and reported using EC syndrome value 0x03.

TACR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the specified registers are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

Note

ACTLR_EL1 is not accessible at EL0.

ACTLR and ACTLR2 are not accessible at EL0.

The Auxiliary Control Registers are IMPLEMENTATION DEFINED registers that might implement global control bits for the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TIDCP, bit [20]

Trap IMPLEMENTATION DEFINED functionality. Traps EL1 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL2, when EL2 is enabled in the current Security state as follows:

- In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped:
  - IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}, and are reported using EC syndrome value 0x18.
  - IMPLEMENTATION DEFINED System instructions, which are accessed using SYSP, with CRn == {11, 15}, and are reported using EC syndrome value 0x14.
  - IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome 0x18.
  - IMPLEMENTATION DEFINED System registers, which are accessed using MRRS and MSRR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome 0x14.
- In AArch32 state, MCR and MRC access to instructions with the following encodings are trapped and reported using EC syndrome value 0x03:
  - All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}.
  - All coproc==p15, CRn==c10, opc1 =={0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}.
  - All coproc==p15, CRn==c11, opc1=={0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}.

When this functionality is accessed from EL0:

- If FEAT_TIDCP1 is implemented and the Effective value of SCTLR_EL1.TIDCP is 1, any accesses from EL0 are trapped to EL1.
- Otherwise, if FEAT_TIDCP1 is implemented and the Effective value of SCTLR_EL2.TIDCP is 1, any accesses from EL0 are trapped to EL2.
- Otherwise:
  - If HCR_EL2.TIDCP is 1, it is IMPLEMENTATION DEFINED whether any accesses from EL0 are trapped to EL2.
  - If HCR_EL2.TIDCP is 0, any accesses from EL0 are UNDEFINED and generate an exception that is taken to EL1 or EL2.

TIDCP	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to or execution of the specified encodings reserved for IMPLEMENTATION DEFINED functionality are trapped to EL2, when EL2 is enabled in the current Security state.

An implementation can also include IMPLEMENTATION DEFINED registers that provide additional controls, to give finer-grained control of the trapping of IMPLEMENTATION DEFINED features.

Note

The trapping of accesses to these registers from EL1 is higher priority than an exception resulting from the register access being UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TSC, bit [19]

Trap SMC instructions. Traps EL1 execution of SMC instructions to EL2, when EL2 is enabled in the current Security state.

If execution is in AArch64 state, the trap is reported using EC syndrome value 0x17.

If execution is in AArch32 state, the trap is reported using EC syndrome value 0x13.

Note

HCR_EL2.TSC traps execution of the SMC instruction. It is not a routing control for the SMC exception. Trap exceptions and SMC exceptions have different preferred return addresses.

TSC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	If EL3 is implemented, then any attempt to execute an SMC instruction at EL1 is trapped to EL2, when EL2 is enabled in the current Security state, regardless of the value of SCR_EL3.SMD. If EL3 is not implemented and the Effective value of HCR_EL2.NV is 1, then any attempt to execute an SMC instruction at EL1 using AArch64 is trapped to EL2. If EL3 is not implemented and the Effective value of HCR_EL2.NV is 0, then it is IMPLEMENTATION DEFINED whether: • Any attempt to execute an SMC instruction at EL1 is trapped to EL2, when EL2 is enabled in the current Security state. • Any attempt to execute an SMC instruction is UNDEFINED.

In AArch32 state, the Armv8-A architecture permits, but does not require, this trap to apply to conditional SMC instructions that fail their condition code check, in the same way as with traps on other conditional instructions.

SMC instructions are UNDEFINED at EL0.

If EL3 is not implemented, and the Effective value of HCR_EL2.NV is 0, then it is IMPLEMENTATION DEFINED whether this bit is:

- RES0.
- Implemented with the functionality as described in HCR_EL2.TSC.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TID3, bit [18]

Trap ID group 3. Traps EL1 reads of group 3 ID registers to EL2, when EL2 is enabled in the current Security state, as follows:

In AArch64 state:

- Reads of the following registers are trapped to EL2, reported using EC syndrome value 0x18:
  - ID_PFR0_EL1, ID_PFR1_EL1, ID_DFR0_EL1, ID_AFR0_EL1, ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, ID_ISAR5_EL1, MVFR0_EL1, MVFR1_EL1, and MVFR2_EL1.
  - ID_AA64PFR0_EL1, ID_AA64PFR1_EL1, ID_AA64DFR0_EL1, ID_AA64DFR1_EL1, ID_AA64ISAR0_EL1, ID_AA64ISAR1_EL1, ID_AA64MMFR0_EL1, ID_AA64MMFR1_EL1, ID_AA64AFR0_EL1, and ID_AA64AFR1_EL1.
  - If FEAT_FGT is implemented, reads of the following registers are trapped to EL2. If FEAT_FGT is not implemented, reads of the following registers are trapped to EL2, unless the registers are implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses are trapped to EL2.
    - ID_PFR2_EL1, ID_MMFR4_EL1 and ID_MMFR5_EL1.
    - ID_AA64MMFR3_EL1.
    - ID_AA64MMFR4_EL1.
    - ID_AA64PFR2_EL1.
    - ID_AA64MMFR2_EL1 and ID_ISAR6_EL1.

            - ID_DFR1_EL1.
            - ID_AA64ZFR0_EL1.
            - ID_AA64SMFR0_EL1.
            - ID_AA64ISAR2_EL1.
        - If FEAT_FGT is implemented, this field traps all MRS accesses to registers in the following range that are not already mentioned in this field description: op0 == 3, op1 == 0, CRn == 0, CRm == {2-7}, op2 == {0-7}. If FEAT_FGT is not implemented, it is IMPLEMENTATION DEFINED whether this field traps accesses to registers in the range.

In AArch32 state:

- VMRS access to MVFR0, MVFR1, and MVFR2, are trapped to EL2, reported using EC syndrome value `0x08`, unless access is also trapped by HCPTR which takes priority.
- MRC access to the following registers are trapped to EL2, reported using EC syndrome value `0x03`:
    - ID_PFR0, ID_PFR1, ID_PFR2, ID_DFR0, ID_AFR0, ID_MMFR0, ID_MMFR1, ID_MMFR2, ID_MMFR3, ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, ID_ISAR4, and ID_ISAR5.
    - If FEAT_FGT is implemented:
        - ID_MMFR4 and ID_MMFR5.
        - ID_ISAR6.
        - ID_DFR1.
        - This field traps all MRC accesses to encodings in the following range that are not already mentioned in this field description: coproc == p15, opc1 == 0, CRn == c0, CRm == {c2-c7}, opc2 == {0-7}.
    - If FEAT_FGT is not implemented:
        - ID_MMFR4 and ID_MMFR5 are trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to ID_MMFR4 or ID_MMFR5 are trapped.
        - ID_ISAR6 is trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to ID_ISAR6 are trapped to EL2.
        - ID_DFR1 is trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to ID_DFR1 are trapped to EL2.
        - Otherwise, it is IMPLEMENTATION DEFINED whether this bit traps all MRC accesses to registers in the following range not already mentioned in this field description with coproc == p15, opc1 == 0, CRn == c0, CRm == {c2-c7}, opc2 == {0-7}.

TID3	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	The specified EL1 read accesses to ID group 3 registers are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TID2, bit [17]

Trap ID group 2. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64, reads of CTR_EL0, CCSIDR_EL1, CCSIDR2_EL1, CLIDR_EL1, and CSSELR_EL1 are trapped to EL2, reported using EC syndrome value `0x18`.
- If EL0 is using AArch64 and the value of SCTLR_EL1.UCT is not 0, reads of CTR_EL0 are trapped to EL2, reported using EC syndrome value `0x18`. If the value of SCTLR_EL1.UCT is 0, then EL0 reads of CTR_EL0 are trapped to EL1 and the resulting exception takes precedence over this trap.
- If EL1 is using AArch64, writes to CSSELR_EL1 are trapped to EL2, reported using EC syndrome value `0x18`.
- If EL1 is using AArch32, reads of CTR, CCSIDR, CCSIDR2, CLIDR, and CSSELR are trapped to EL2, reported using EC syndrome value `0x03`.
- If EL1 is using AArch32, writes to CSSELR are trapped to EL2, reported using EC syndrome value `0x03`.

TID2	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	The specified EL1 and EL0 accesses to ID group 2 registers are trapped to EL2, when EL2 is enabled in the current Security state.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TID1, bit [16]

Trap ID group 1. Traps EL1 reads of the following registers to EL2, when EL2 is enabled in the current Security state as follows:

- In AArch64 state, accesses of REVIDR_EL1, AIDR_EL1, and SMIDR_EL1, reported using EC syndrome value 0x18.
- In AArch32 state, accesses of TCMTR, TLBTR, REVIDR, and AIDR, reported using EC syndrome value 0x03.

TID1	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL1 read accesses to ID group 1 registers are trapped to EL2, when EL2 is enabled in the current Security state.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TID0, bit [15]

When FEAT_AA32 is implemented:

Trap ID group 0. Traps the following register accesses to EL2:

- EL1 reads of the JIDR, reported using EC syndrome value 0x05.
- If the JIDR is RAZ from EL0, EL0 reads of the JIDR, reported using EC syndrome value 0x05.
- EL1 accesses using VMRS of the FPSID, reported using EC syndrome value 0x08.

Note

- It is IMPLEMENTATION DEFINED whether the JIDR is RAZ or UNDEFINED at EL0. If it is UNDEFINED at EL0, then any resulting exception takes precedence over this trap.
- The FPSID is not accessible at EL0 using AArch32.
- Writes to the FPSID are ignored, and not trapped by this control.

TID0	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL1 read accesses to ID group 0 registers are trapped to EL2, when EL2 is enabled in the current Security state.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWE, bit [14]

Traps EL0 and EL1 execution of WFE instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states, reported using EC syndrome value 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

TWE	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFE instruction at EL0 or EL1 is trapped to EL2, when EL2 is enabled in the current Security state, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE or SCTLR_EL1.nTWE.

In AArch32 state, the attempted execution of a conditional WFE instruction is trapped only if the instruction passes its condition code check.

Note

Since a WFE can complete at any time, even without a Wakeup event, the traps on WFE are not guaranteed to be taken, even if the WFE is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

For more information about when WFE instructions can cause the PE to enter a low-power state, see 'Wait for Event mechanism and Send event'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TWI, bit [13]

Traps EL0 and EL1 execution of WFI instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states, reported using EC syndrome value `0x01`.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

TWI	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Any attempt to execute a WFI instruction at EL0 or EL1 is trapped to EL2, when EL2 is enabled in the current Security state, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI or SCTLR_EL1.nTWI.

In AArch32 state, the attempted execution of a conditional WFI instruction is trapped only if the instruction passes its condition code check.

Note

Since a WFI can complete at any time, even without a Wakeup event, the traps on WFI are not guaranteed to be taken, even if the WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

For more information about when WFI instructions can cause the PE to enter a low-power state, see 'Wait for Interrupt'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DC, bit [12]

Default Cacheability.

DC	Meaning
`0b0`	This control has no effect on the EL1&0 translation regime.
`0b1`	In any Security state: • When EL1 is using AArch64, the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of SCTLR_EL1. • When EL1 is using AArch32, the PE behaves as if the value of the SCTLR.M field is 0 for all purposes other than returning the value of a direct read of SCTLR. • The PE behaves as if the value of the HCR_EL2.VM field is 1 for all purposes other than returning the value of a direct read of HCR_EL2. • The memory type produced by stage 1 of the EL10 translation regime is Normal Non-Shareable, Inner Write-Back Read-Allocate Write-Allocate, Outer Write-Back Read-Allocate Write-Allocate.

This field has no effect on the EL2, EL2&0, and EL3 translation regimes.

This bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BSU, bits [11:10]

Barrier Shareability upgrade. This field determines the minimum shareability domain that is applied to any barrier instruction executed from EL1 or EL0:

BSU	Meaning
0b00	No effect.
0b01	Inner Shareable.
0b10	Outer Shareable.
0b11	Full system.

This value is combined with the specified level of the barrier held in its instruction, using the same principles as combining the shareability attributes from two stages of address translation.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FB, bit [9]

Force broadcast. Causes the following Non-shareable invalidate instructions to be broadcast within the Inner Shareable domain when executed from EL1:

- In AArch32 state: BPIALL, TLBIALL, TLBIMVA, TLBIASID, DTLBIALL, DTLBIMVA, DTLBIASID, ITLBIALL, ITLBIMVA, ITLBIASID, TLBIMVAA, ICIALLU, TLBIMVAL, and TLBIMVAAL.
- In AArch64 state:
  - All `TLBI` instructions that are executable at EL1 and do not specify IS or OS.
  - All `TLBIP` instructions that are executable at EL1 and do not specify IS or OS.
  - `IC IALLU`.

For more information, see 'A64 System instructions for TLB maintenance'.

FB	Meaning
0b0	The specified instructions are not affected by this control.
0b1	When one of the specified Non-shareable instructions is executed at EL1, the operation is broadcast within the Inner Shareable shareability domain.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VSE, bit [8]

Virtual SError exception.

VSE	Meaning
0b0	This mechanism is not making a virtual SError exception pending.
0b1	A virtual SError exception is pending because of this mechanism.

The virtual SError exception is enabled only when HCR_EL2.TGE is 0 and either HCR_EL2.AMO is 1 or FEAT_DoubleFault2 is implemented and the Effective value of HCRX_EL2.TMEA is 1.

When FEAT_E3DSE is implemented, virtual SError exceptions pended by this field have priority over delegated SError exceptions pended by SCR_EL3.DSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VI, bit [7]

Virtual IRQ Interrupt.

VI	Meaning
0b0	This mechanism is not making a virtual IRQ pending.
0b1	A virtual IRQ is pending because of this mechanism.

The virtual IRQ is enabled only when the value of HCR_EL2.{TGE, IMO} is {0, 1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VF, bit [6]

Virtual FIQ Interrupt.

VF	Meaning
0b0	This mechanism is not making a virtual FIQ pending.
0b1	A virtual FIQ is pending because of this mechanism.

The virtual FIQ is enabled only when the value of HCR_EL2.{TGE, FMO} is {0, 1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AMO, bit [5]

Physical SError exception routing.

AMO	Meaning
0b0	Physical SError exceptions are unaffected by this mechanism. That is, physical SError exceptions are not taken to EL2 unless routed to EL2 by another control. Virtual SError exceptions are not enabled by this mechanism.
0b1	When executing at any Exception level, and EL2 is enabled in the current Security state, all of the following apply: • Physical SError exceptions are taken to EL2, unless they are routed to EL3. • If FEAT_E3DSE is implemented then delegated SError exceptions enabled by SCR_EL3.DSE are taken to EL2. • If HCR_EL2.TGE is 0 then virtual SError exceptions are enabled.

If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:

- Regardless of the value of HCR_EL2.AMO, physical SError exceptions target EL2 unless they are routed to EL3.
- If FEAT_E3DSE is implemented then regardless of the value of HCR_EL2.AMO, delegated SError exceptions target EL2.
- When the Effective value of HCR_EL2.E2H is not 1, the Effective value of this field is 1.
- When the Effective value of HCR_EL2.E2H is 1, the Effective value of this field is 0.

For more information, see 'Asynchronous exception routing'.

Virtual SError exceptions are disabled when the Effective value of HCR_EL2.AMO is 0 and either FEAT_DoubleFault2 is not implemented or the Effective value of HCRX_EL2.TMEA is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMO, bit [4]

Physical IRQ Routing.

IMO	Meaning
0b0	When executing at Exception levels below EL2, and EL2 is enabled in the current Security state: • When the value of HCR_EL2.TGE is 0, Physical IRQ interrupts are not taken to EL2. • When the value of HCR_EL2.TGE is 1, Physical IRQ interrupts are taken to EL2 unless they are routed to EL3. • Virtual IRQ interrupts are disabled.
0b1	When executing at any Exception level, and EL2 is enabled in the current Security state: • Physical IRQ interrupts are taken to EL2, unless they are routed to EL3. • When the value of HCR_EL2.TGE is 0, then Virtual IRQ interrupts are enabled.

If EL2 is enabled in the current Security state, and the value of HCR_EL2.TGE is 1:

- Regardless of the value of the IMO bit, physical IRQ Interrupts target EL2 unless they are routed to EL3.
- When the Effective value of HCR_EL2.E2H is not 1, the Effective value of this field is 1.
- When the Effective value of HCR_EL2.E2H is 1, the Effective value of this field is 0.

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FMO, bit [3]

Physical FIQ Routing.

FMO	Meaning
0b0	When executing at Exception levels below EL2, and EL2 is enabled in the current Security state: • When the value of HCR_EL2.TGE is 0, Physical FIQ interrupts are not taken to EL2. • When the value of HCR_EL2.TGE is 1, Physical FIQ interrupts are taken to EL2 unless they are routed to EL3. • Virtual FIQ interrupts are disabled.
0b1	When executing at any Exception level, and EL2 is enabled in the current Security state: • Physical FIQ interrupts are taken to EL2, unless they are routed to EL3. • When HCR_EL2.TGE is 0, then Virtual FIQ interrupts are enabled.

If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:

- Regardless of the value of the FMO bit, physical FIQ Interrupts target EL2 unless they are routed to EL3.
- When the Effective value of HCR_EL2.E2H is not 1, the Effective value of this field is 1.
- When the Effective value of HCR_EL2.E2H is 1, the Effective value of this field is 0.

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PTW, bit [2]

Protected Table Walk. In the EL1&0 translation regime, a translation table access made as part of a stage 1 translation table walk is subject to a stage 2 translation. The combining of the memory type attributes from the two stages of translation means the access might be made to a type of Device memory. If this occurs, then the value of this bit determines the behavior:

PTW	Meaning
0b0	The translation table walk occurs as if it is to Normal Non-cacheable memory. This means it can be made speculatively.
0b1	The memory access generates a stage 2 Permission fault.

This bit is permitted to be cached in a TLB.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SWIO, bit [1]

Set/Way Invalidation Override. Causes EL1 execution of the data cache invalidate by set/way instructions to perform a data cache clean and invalidate by set/way:

SWIO	Meaning
0b0	This control has no effect on the operation of data cache invalidate by set/way instructions.
0b1	Data cache invalidate by set/way instructions perform a data cache clean and invalidate by set/way.

When the value of this bit is 1:

AArch32: DCISW performs the same invalidation as a DCCISW instruction.

AArch64: DC ISW performs the same invalidation as a DC CISW instruction.

This bit can be implemented as RES1.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VM, bit [0]

Virtualization enable. Enables stage 2 address translation for the EL1&0 translation regime, when EL2 is enabled in the current Security state.

VM	Meaning
0b0	EL1&0 stage 2 address translation disabled.
0b1	EL1&0 stage 2 address translation enabled.

When the value of this bit is 1, data cache invalidate instructions executed at EL1 perform a data cache clean and invalidate. For the invalidate by set/way instruction this behavior applies regardless of the value of the HCR_EL2.SWIO bit.

This bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x078];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = HCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HCR_EL2;
```

```
MSR HCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x078] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HCR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HCRX_EL2, Extended Hypervisor Configuration Register

The HCRX_EL2 characteristics are:

Purpose

Provides configuration controls for virtualization, including defining whether various operations are trapped to EL2.

Configuration

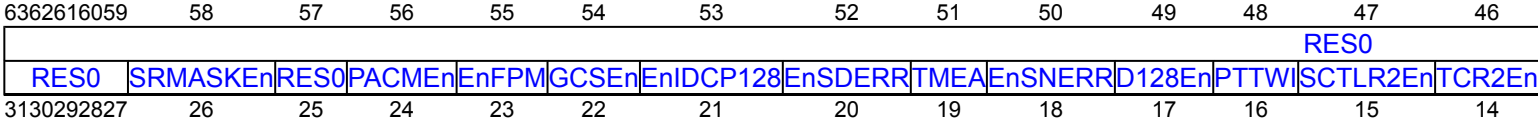
This register is present only when FEAT_HCX is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HCRX_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HCRX_EL2 is a 64-bit register.

Field descriptions



Bits [63:27]

Reserved, RES0.

SRMASKEn, bit [26]

When FEAT_SRMASK is implemented:

If EL1 is using AArch64, accesses to the following registers are trapped to EL2 and reported using EC syndrome value `0x18`:

- CPACRMASK_EL1.
- SCTLRMASK_EL1.
- SCTLR2MASK_EL1.
- TCRMASK_EL1.
- TCR2MASK_EL1.
- ACTLRMASK_EL1, if it is implemented.

SRMASKEn	Meaning
`0b0`	EL1 accesses to the specified EL1 registers are trapped to EL2. The values in the registers are treated as 0.
`0b1`	This control does not cause any instructions to be trapped.

Traps generated by this control have a lower priority than traps generated by the HFGRTR2_EL2 and HFGWTR2_EL2 controls.

When EL2 is not implemented or not enabled in the current Security state, the Effective value of this field is 1.

Otherwise, when the Effective value of SCR_EL3.HXEn is 0, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [25]

Reserved, RES0.

PACMEn, bit [24]
When FEAT_PAuth_LR is implemented:

PACM Enable. Controls the effect of a PACM instruction at EL1 and EL0.

PACMEn	Meaning
0b0	The effects of PACM are disabled at EL1 and EL0, regardless of the value in any other controls.
0b1	This control does not disable the effect of PACM at EL1 and EL0.

The Effective value of this field is 1 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Effective value of this field is 0 when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [23]
When FEAT_FPMR is implemented:

Enables direct and indirect accesses to FPMR from EL0 and EL1.

When accesses to FPMR are disabled by this control:

- Direct accesses to FPMR from EL0 and EL1 are trapped to EL2 and reported using EC syndrome value 0x18.
- Execution of FP8 data-processing instructions that indirectly access FPMR is UNDEFINED at EL0 and EL1.

EnFPM	Meaning
0b0	Direct and indirect accesses to FPMR are disabled at EL1 and EL0.
0b1	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

The Effective value of this field is 1 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Effective value of this field is 0 when all of the following are true:

- EL2 is implemented and enabled in the current Security state.

- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

GCSEn, bit [22]
When FEAT_GCS is implemented:

Guarded Control Stack enable. Controls Guarded Control Stack behavior at EL1 and EL0.

GCSEn	Meaning
0b0	The Guarded Control Stack is disabled at EL1 and EL0.
0b1	Guarded Control Stack behavior at EL1 and EL0 is not affected by mechanism.

The Effective value of this field is 1 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Effective value of this field is 0 when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIDCP128, bit [21]
When FEAT_SYSREG128 is implemented:

Enables access to IMPLEMENTATION DEFINED 128-bit System registers.

EnIDCP128	Meaning
0b0	Accesses at EL1, EL0 to IMPLEMENTATION DEFINED 128-bit System registers are trapped to EL2 using EC syndrome value 0x14, unless the access generates a higher priority exception. Disables the functionality of the 128-bit IMPLEMENTATION DEFINED System registers that are accessible at EL2.
0b1	No accesses are trapped by this control.

The Effective value of this field is 1 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Effective value of this field is 0 when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSDERR, bit [20]
When FEAT_ADERR is implemented:

Enable Synchronous Device Read Error. Override SCTLR2_EL1.EnADERR.

EnSDERR	Meaning
0b0	This field has no effect on External aborts on Device memory reads in the EL1&0 translation regime.
0b1	External abort on Device memory reads generate synchronous Data Abort exceptions in the EL1&0 translation regime.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Setting this field to 1 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Device memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, HCRX_EL2.EnSDERR is 1, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 1 might have a performance impact for Device memory reads.

This field is ignored by the PE and has an Effective value of 0 when any of the following are true:

- All of the following are true:
  - FEAT_ANERR is implemented.
  - ID_AA64MMFR3_EL1.ADERR reads as 0b0010.
  - HCRX_EL2.{EnSDERR, EnSNERR} == {1, 0}.
- The Effective value of SCR_EL3.HXEn is 0.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.
- EL2 is not implemented or not enabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMEA, bit [19]
When FEAT_DoubleFault2 is implemented:

Trap Masked External Aborts. Controls whether a masked error exception at a lower Exception level is taken to EL2.

TMEA	Meaning
0b0	Synchronous External abort exceptions, physical SError exceptions, and delegated SError exceptions at EL1 and EL0 are unaffected by this mechanism. That is, these exceptions are not taken to EL2 unless routed to EL2 by another control. Virtual SError exceptions are not enabled by this mechanism.
0b1	When executing at Exception levels below EL2, if EL2 is enabled in the current Security state, then all of the following apply: • When PSTATE.A is 1, synchronous External abort exceptions are taken to EL2, unless they are routed to EL3. • Masked physical SError exceptions are taken to EL2, unless they are routed to EL3. • If FEAT_E3DSE is implemented, then masked delegated SError exceptions enabled by SCR_EL3.DSE are taken to EL2. • If HCR_EL2.TGE is 0, then virtual SError exceptions are enabled.

The Effective value of this field is 0 when any of the following are true:

- The Effective value of SCR_EL3.HXEn is 0.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.
- EL2 is not implemented or not enabled in the current Security state.

Virtual SError exceptions are disabled when the Effective value of HCR_EL2.AMO is 0 and the Effective value of HCRX_EL2.TMEA is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSNERR, bit [18]
When FEAT_ANERR is implemented:

Enable Synchronous Normal Read Error. Override SCTLR2_EL1.EnANERR.

EnSNERR	Meaning
0b0	This field has no effect on External aborts on Normal memory reads in the EL1&0 translation regime.
0b1	External abort on Normal memory reads generate synchronous Data Abort exceptions in the EL1&0 translation regime.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Setting this field to 1 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Normal memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, HCRX_EL2.EnSNERR is 1, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 1 might have a performance impact for Normal memory reads.

This field is ignored by the PE and has an Effective value of 0 when any of the following are true:

- All of the following are true:
  - FEAT_ADERR is implemented.
  - ID_AA64MMFR3_EL1.ANERR reads as 0b0010.
  - HCRX_EL2.{EnSDERR, EnSNERR} == {0, 1}.
- The Effective value of SCR_EL3.HXEn is 0.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.
- EL2 is not implemented or not enabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D128En, bit [17]
When FEAT_D128 is implemented:

128-bit System Register trap control. Enable access to 128-bit System Registers that relate to VMSAv9-128 via MRRS, MSRR instructions.

If EL1 is using AArch64, accesses to the following registers are trapped to EL2 and reported using EC syndrome value 0x14:

- TTBR0_EL1.
- TTBR1_EL1.
- If FEAT_THE is implemented, RCWMASK_EL1, RCWSMASK_EL1.
- PAR_EL1.

D128En	Meaning
0b0	EL1 accesses to the specified registers are disabled, and trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

When EL2 is not implemented or not enabled in the current Security state, the Effective value of this field is 1.

Otherwise, when the Effective value of SCR_EL3.HXEn is 0, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [16]
When FEAT_THE is implemented:

Permit Translation Table Walk Incoherence.

Permits RCWS instructions to generate writes that have the Reduced Coherence property.

PTTWI	Meaning
0b0	Write accesses generated by RCWS at EL1 and EL0 do not have the Reduced Coherence property.
0b1	Write accesses generated by RCWS at EL1 and EL0 have the Reduced Coherence property, if enabled by TCR2_EL1.PTTWI.

The Effective value of this field is 1 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Effective value of this field is 0 when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The Effective value of SCR_EL3.HXEn is 0.

This field is permitted to be cached in TLB.

This field is permitted to be implemented as a read-only field with a fixed value of 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCTLR2En, bit [15]

When FEAT_SCTLR2 is implemented:

SCTLR2_EL1 Enable. In AArch64 state, accesses to SCTLR2_EL1 are trapped to EL2 and reported using EC syndrome value 0x18.

SCTLR2En	Meaning
0b0	Accesses to SCTLR2_EL1 at EL1 are trapped to EL2, unless the access generates a higher priority exception. The value in SCTLR2_EL1 is treated as 0.
0b1	This control does not cause any instructions to be trapped.

When EL2 is not implemented or not enabled in the current Security state, the Effective value of this field is 1.

Otherwise, when the Effective value of SCR_EL3.HXEn is 0, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCR2En, bit [14]

When FEAT_TCR2 is implemented:

TCR2_EL1 Enable. In AArch64 state, accesses to TCR2_EL1 are trapped to EL2 and reported using EC syndrome value 0x18.

TCR2En	Meaning
0b0	Accesses to TCR2_EL1 at EL1 are trapped to EL2, unless the access generates a higher priority exception. The value in TCR2_EL1 is treated as 0.
0b1	This control does not cause any instructions to be trapped.

When EL2 is not implemented or not enabled in the current Security state, the Effective value of this field is 1.

Otherwise, when the Effective value of SCR_EL3.HXEn is 0, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [13:12]

Reserved, RES0.

MSCEn, bit [11]
When FEAT_MOPS is implemented:

Memory Set and Memory Copy instructions Enable. Enables execution of the CPY*, SETG*, SETP*, SETM*, and SETE* instructions at EL1 or EL0.

MSCEn	Meaning
0b0	Execution of the Memory Copy and Memory Set instructions is UNDEFINED at EL1 or EL0.
0b1	This control does not cause any instructions to be UNDEFINED.

The Effective value of this field is 1 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Effective value of this field is 0 when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
    - When the highest implemented Exception level is EL2, this field resets to '0'.
    - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MCE2, bit [10]
When FEAT_MOPS is implemented:

Controls Memory Copy and Memory Set exceptions generated as part of attempting to execute the Memory Copy and Memory Set instructions from EL1.

MCE2	Meaning
0b0	Memory Copy and Memory Set exceptions generated from EL1 are taken to EL1.
0b1	Memory Copy and Memory Set exceptions generated from EL1 are taken to EL2.

The Effective value of this field is 0 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
    - When the highest implemented Exception level is EL2, this field resets to '0'.
    - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [9]
When FEAT_CMOW is implemented:

Controls cache maintenance instruction permission for the following instructions executed at EL1 or EL0.

- IC IVAU and DC CIVAC.
- If FEAT_MTE is implemented, DC CIGDVAC and DC CIGVAC.
- If FEAT_PoPS is implemented, DC CIVAPS.
- If FEAT_PoPS and FEAT_MTE2 are implemented, DC CIGDVAPS.
- ICIMVAU and DCCIMVAC.

CMOW	Meaning
0b0	This control does not cause any instructions to generate a stage 2 Permission fault.
0b1	Unless the instruction generates a higher priority exception, EL1 and EL0 execution of the specified instructions without stage 2 write permission generates a stage 2 Permission fault.

For this control, stage 2 has write permission if S2AP[1] is 1 or DBM is 1 in the stage 2 descriptor. The instructions do not cause an update to the dirty state.

The Effective value of this field is 0 when any of the following are true:

- The Effective value of SCR_EL3.HXEn is 0.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.
- EL2 is not implemented or not enabled in the current Security state.

This field is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VFNMI, bit [8]
When FEAT_NMI is implemented:

Virtual FIQ Interrupt with Superpriority. Enables signaling of virtual FIQ interrupts with Superpriority.

VFNMI	Meaning
0b0	When HCR_EL2.VF is 1, a signaled pending virtual FIQ interrupt does not have Superpriority.
0b1	When HCR_EL2.VF is 1, a signaled pending virtual FIQ interrupt has Superpriority.

When HCR_EL2.VF is 0, this field has no effect.

The Effective value of this field is 0 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VINMI, bit [7]

When FEAT_NMI is implemented:

Virtual IRQ Interrupt with Superpriority. Enables signaling of virtual IRQ interrupts with Superpriority.

VINMI	Meaning
0b0	When HCR_EL2.VI is 1, a signaled pending virtual IRQ interrupt does not have Superpriority.
0b1	When HCR_EL2.VI is 1, a signaled pending virtual IRQ interrupt has Superpriority.

When HCR_EL2.VI is 0, this field has no effect.

The Effective value of this field is 0 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TALLINT, bit [6]

When FEAT_NMI is implemented:

Traps the following writes at EL1 using AArch64 to EL2, when EL2 is implemented and enabled:

- MSR (register) writes of ALLINT.
- MSR (immediate) writes of ALLINT with a value of 1.

TALLINT	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified MSR accesses at EL1 using AArch64 are trapped to EL2.

The Effective value of this field is 0 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SMPME, bit [5]

When FEAT_SME is implemented:

Streaming Mode Priority Mapping Enable.

Controls mapping of the value of SMPRI_EL1.Priority for streaming execution priority at EL0 or EL1.

SMPME	Meaning
0b0	The effective priority value is taken from SMPRI_EL1.Priority.
0b1	The effective priority value is: • When the current Exception level is EL2 or EL3, the value of SMPRI_EL1.Priority. • When the current Exception level is EL0 or EL1, the value of the SMPRIMAP_EL2 field corresponding to the value of SMPRI_EL1.Priority.

When SMIDR_EL1.SMPS is '0', this field is RES0.

The Effective value of this field is 0 when any of the following are true:

- The Effective value of SCR_EL3.HXEn is 0.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.
- EL2 is not implemented or not enabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FGTnXS, bit [4]

When FEAT_XS is implemented:

Determines if the fine-grained traps in HFGITR_EL2 that apply to each of the TLBI maintenance instructions that are accessible at EL1 also apply to the corresponding TLBI maintenance instructions with the nXS qualifier.

FGTnXS	Meaning
0b0	The fine-grained trap in the HFGITR_EL2 that applies to a TLBI maintenance instruction at EL1 also applies to the corresponding TLBI instruction with the nXS qualifier at EL1.
0b1	The fine-grained trap in the HFGITR_EL2 that applies to a TLBI maintenance instruction at EL1 does not apply to the corresponding TLBI instruction with the nXS qualifier at EL1.

The Effective value of this field is 0 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnXS, bit [3]

When FEAT_XS is implemented:

Determines the behavior of TLBI instructions affected by the XS attribute.

This control field also determines whether an AArch64 DSB instruction behaves as a DSB instruction with an nXS qualifier when executed at EL0 and EL1.

FnXS	Meaning
0b0	This control does not have any effect on the behavior of the TLBI maintenance instructions.
0b1	A TLBI maintenance instruction without the nXS qualifier executed at EL1 behaves in the same way as the corresponding TLBI maintenance instruction with the nXS qualifier. An AArch64 DSB instruction that applies to both loads and stores executed at EL1 or EL0 behaves in the same way as the corresponding DSB instruction with the nXS qualifier executed at EL1 or EL0.

The Effective value of this field is 0 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of SCR_EL3.HXEn is 0.

This field is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnASR, bit [2]

When FEAT_LS64_V is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps execution of an ST64BV instruction at EL0 or EL1 to EL2.

EnASR	Meaning
0b0	Execution of an ST64BV instruction at EL0 is trapped to EL2 if the execution is not trapped by SCTLR_EL1.EnASR. Execution of an ST64BV instruction at EL1 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000000.

The Effective value of this field is 1 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Effective value of this field is 0 when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnALS, bit [1]
When FEAT_LS64 is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps execution of an LD64B or ST64B instruction at EL0 or EL1 to EL2.

EnALS	Meaning
0b0	Execution of an LD64B or ST64B instruction at EL0 is trapped to EL2 if the execution is not trapped by SCTLR_EL1.EnALS. Execution of an LD64B or ST64B instruction at EL1 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

A trap of an LD64B or ST64B instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000002.

The Effective value of this field is 1 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Effective value of this field is 0 when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
    - When the highest implemented Exception level is EL2, this field resets to '0'.
    - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [0]
When FEAT_LS64_ACCDATA is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps execution of an ST64BV0 instruction at EL0 or EL1 to EL2.

EnAS0	Meaning
0b0	Execution of an ST64BV0 instruction at EL0 is trapped to EL2 if the execution is not trapped by SCTLR_EL1.EnAS0. Execution of an ST64BV0 instruction at EL1 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV0 instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000001.

The Effective value of this field is 1 when any of the following are true:

- EL2 is not implemented or not enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Effective value of this field is 0 when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The Effective value of SCR_EL3.HXEn is 0.

The reset behavior of this field is:

- On a Warm reset:
    - When the highest implemented Exception level is EL2, this field resets to '0'.
    - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing HCRX_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HCRX_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_HCX) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x0A0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.HXEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HXEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HCRX_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HCRX_EL2;
```

MSR HCRX_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_HCX) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x0A0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.HXEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HXEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HCRX_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HCRX_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDBSSBR_EL2, Hardware Dirty State Tracking Structure Base Register

The HDBSSBR_EL2 characteristics are:

Purpose

Control register for HDBSS base address.

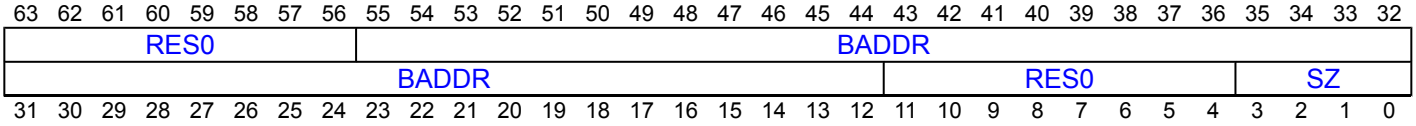
Configuration

This register is present only when FEAT_HDBSS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HDBSSBR_EL2 are UNDEFINED.

Attributes

HDBSSBR_EL2 is a 64-bit register.

Field descriptions



Bits [63:56]

Reserved, RES0.

BADDR, bits [55:12]

HDBSS base address, bits [55:12].

- Bits [55:12] of the base address are the value of this field.
- Bits [11:0] of the base address are 0.

Bits of this field above the implemented physical address size, indicated in ID_AA64MMFR0_EL1.PARange, are RES0.

Based on the value of the SZ field of this register, for encodings of the SZ field greater than 4KB, bits [(SZ+12-1):12] of this field are RES0 such that the base address of the HDBSS is aligned to its size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:4]

Reserved, RES0.

SZ, bits [3:0]

Size of the HDBSS.

SZ	Meaning
0b0000	4KB
0b0001	8KB
0b0010	16KB
0b0011	32KB
0b0100	64KB
0b0101	128KB
0b0110	256KB
0b0111	512KB
0b1000	1MB
0b1001	2MB

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HDBSSBR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HDBSSBR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_HDBSS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x2E0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.HDBSSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HDBSSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HDBSSBR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HDBSSBR_EL2;
```

MSR HDBSSBR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_HDBSS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x2E0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.HDBSSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HDBSSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HDBSSBR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HDBSSBR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDBSSPROD_EL2, Hardware Dirty State Tracking Structure Producer Register

The HDBSSPROD_EL2 characteristics are:

Purpose

Allows producer to update write index for HDBSS.

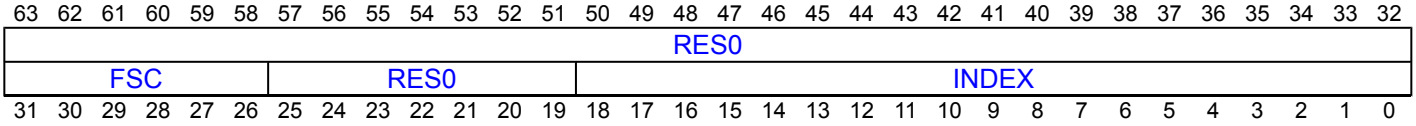
Configuration

This register is present only when FEAT_HDBSS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HDBSSPROD_EL2 are UNDEFINED.

Attributes

HDBSSPROD_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

FSC, bits [31:26]

Fault Status Code.

FSC	Meaning	Applies when
0b000000	The PE has not experienced any error on writes to the HDBSS.	
0b010000	External Abort on write to HDBSS.	
0b101000	Granule Protection Fault on write to HDBSS.	When FEAT_RME is implemented

If this field is non-zero then one or more HDBSS writes have experienced a fault since this field was last set to zero, and the associated HDBSS entries have been lost.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:19]

Reserved, RES0.

INDEX, bits [18:0]

This field indicates the index of the HDBSS entry that will be written to next.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HDBSSPROD_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HDBSSPROD_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0011	0b011

```
if !(IsFeatureImplemented(FEAT_HDBSS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x300];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.HDBSSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HDBSSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HDBSSPROD_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HDBSSPROD_EL2;
```

```
MSR HDBSSPROD_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0011	0b011

```
if !(IsFeatureImplemented(FEAT_HDBSS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x300] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.HDBSSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HDBSSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HDBSSPROD_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HDBSSPROD_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDFGRTR2_EL2, Hypervisor Debug Fine-Grained Read Trap Register 2

The HDFGRTR2_EL2 characteristics are:

Purpose

Provides controls for traps of `MRS` and `MRC` reads of debug, trace, PMU, and Statistical Profiling System registers.

Configuration

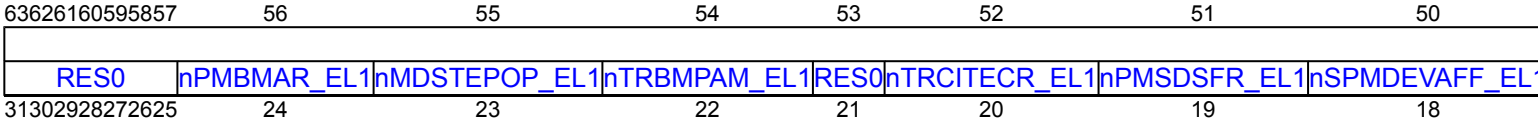
This register is present only when FEAT_FGT2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HDFGRTR2_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HDFGRTR2_EL2 is a 64-bit register.

Field descriptions



Bits [63:25]

Reserved, RES0.

nPMBMAR_EL1, bit [24]

When FEAT_SPE_nVM is implemented:

Trap `MRS` reads of PMBMAR_EL1 at EL1 using AArch64 to EL2.

nPMBMAR_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then `MRS` reads of PMBMAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.
`0b1`	`MRS` reads of PMBMAR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nMDSTEPOP_EL1, bit [23]

When FEAT_STEP2 is implemented:

Trap MRS reads of MDSTEPOP_EL1 at EL1 using AArch64 to EL2.

nMDSTEPOP_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of MDSTEPOP_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of MDSTEPOP_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTRBMPAM_EL1, bit [22]

When FEAT_TRBE_MPAM is implemented:

Trap MRS reads of TRBMPAM_EL1 at EL1 using AArch64 to EL2.

nTRBMPAM_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of TRBMPAM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of TRBMPAM_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [21]

Reserved, RES0.

nTRCITECR_EL1, bit [20]

When FEAT_ITE is implemented:

Trap MRS reads of TRCITECR_EL1 at EL1 using AArch64 to EL2.

nTRCITECR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of TRCITECR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of TRCITECR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMSDSFR_EL1, bit [19]

When FEAT_SPE_FDS is implemented:

Trap MRS reads of PMSDSFR_EL1 at EL1 using AArch64 to EL2.

nPMSDSFR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of PMSDSFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of PMSDSFR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMDEVAFF_EL1, bit [18]

When FEAT_SPMU is implemented:

Trap MRS reads of SPMDEVAFF_EL1 at EL1 using AArch64 to EL2.

nSPMDEVAFF_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of SPMDEVAFF_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SPMDEVAFF_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMID, bit [17]

When FEAT_SPMU is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- SPMCFGR_EL1.
- SPMCGCR<n>_EL1.
- SPMDEVARCH_EL1.
- SPMIIDR_EL1.

nSPMID	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMSCR_EL1, bit [16]

When FEAT_SPMU is implemented:

Trap MRS reads of SPMSCR_EL1 at EL1 using AArch64 to EL2.

nSPMSCR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of SPMSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SPMSCR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMACCESSR_EL1, bit [15]

When FEAT_SPMU is implemented:

Trap MRS reads of SPMACCESSR_EL1 at EL1 using AArch64 to EL2.

nSPMACCESSR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of SPMACCESSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SPMACCESSR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMCR_EL0, bit [14]

When FEAT_SPMU is implemented:

Trap MRS reads of SPMCR_EL0 at EL1 and EL0 using AArch64 to EL2.

nSPMCR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MRS reads of SPMCR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SPMCR_EL0 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMOVS, bit [13]

When FEAT_SPMU is implemented:

Trap MRS reads at EL1 and EL0 using AArch64 of any of the following AArch64 System registers to EL2:

- SPMOVSCLR_EL0.
- SPMOVSSET_EL0.

nSPMOVS	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MRS reads at EL1 and EL0 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMINTEN, bit [12]

When FEAT_SPMU is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- SPMINTENCLR_EL1.
- SPMINTENSET_EL1.

nSPMINTEN	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMCNTEN, bit [11]

When FEAT_SPMU is implemented:

Trap MRS reads at EL1 and EL0 using AArch64 of any of the following AArch64 System registers to EL2:

- SPMCNTENCLR_EL0.
- SPMCNTENSET_EL0.

nSPMCNTEN	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MRS reads at EL1 and EL0 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMSELR_EL0, bit [10]

When FEAT_SPMU is implemented:

Trap MRS reads of SPMSELR_EL0 at EL1 and EL0 using AArch64 to EL2.

nSPMSELR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MRS reads of SPMSELR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SPMSELR_EL0 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMEVTYPERn_EL0, bit [9]
When FEAT_SPMU is implemented:

Trap MRS reads at EL1 and EL0 using AArch64 of any of the following AArch64 System registers to EL2:

- SPMEVTYPER<n>_EL0.
- SPMEVFILTR<n>_EL0.
- SPMEVFILT2R<n>_EL0.

nSPMEVTYPERn_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MRS reads at EL1 and EL0 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMEVCNTRn_EL0, bit [8]
When FEAT_SPMU is implemented:

Trap MRS reads of SPMEVCNTR<n>_EL0 at EL1 and EL0 using AArch64 to EL2.

nSPMEVCNTRn_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MRS reads of SPMEVCNTR<n>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SPMEVCNTR<n>_EL0 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMSSCR_EL1, bit [7]
When FEAT_PMUv3_SS is implemented:

Trap MRS reads of PMSSCR_EL1 at EL1 using AArch64 to EL2.

nPMSSCR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of PMSSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of PMSSCR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMSSDATA, bit [6]
When FEAT_PMUv3_SS is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- PMCCNTSVR_EL1.
- PMEVCNTSVR<n>_EL1.
- PMICNTSVR_EL1, if FEAT_PMUv3_ICNTR is implemented.

nPMSSDATA	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nMDSELR_EL1, bit [5]

When FEAT_Debugv8p9 is implemented:

Trap MRS reads of MDSELR_EL1 at EL1 using AArch64 to EL2.

nMDSELR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of MDSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of MDSELR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMUACR_EL1, bit [4]

When FEAT_PMUv3p9 is implemented:

Trap MRS reads of PMUACR_EL1 at EL1 using AArch64 to EL2.

nPMUACR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of PMUACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of PMUACR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMICFILTR_EL0, bit [3]

When FEAT_PMUv3_ICNTR is implemented:

Trap MRS reads of PMICFILTR_EL0 at EL1 and EL0 using AArch64 to EL2.

nPMICFILTR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then: • MRS reads of PMICFILTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. • PMCNTENCLR_EL0.F0, PMCNTENSET_EL0.F0, PMOVSCLR_EL0.F0, and PMOVSSET_EL0.F0 read as zero at EL1 and EL0. • PMINTENCLR_EL1.F0 and PMINTENSET_EL1.F0 read as zero at EL1.
0b1	MRS reads of PMICFILTR_EL0 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMICNTR_EL0, bit [2]

When FEAT_PMUv3_ICNTR is implemented:

Trap MRS reads of PMICNTR_EL0 at EL1 and EL0 using AArch64 to EL2.

nPMICNTR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MRS reads of PMICNTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of PMICNTR_EL0 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMIAR_EL1, bit [1]

When FEAT_SEBEP is implemented:

Trap MRS reads of PMIAR_EL1 at EL1 using AArch64 to EL2.

nPMIAR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of PMIAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of PMIAR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMECR_EL1, bit [0]

When FEAT_EBEP is implemented or FEAT_PMUv3_SS is implemented:

Trap MRS reads of PMECR_EL1 at EL1 using AArch64 to EL2.

nPMECR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of PMECR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of PMECR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing HDFGRTR2_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HDFGRTR2_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1A0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HDFGRTR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HDFGRTR2_EL2;
```

MSR HDFGRTR2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1A0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HDFGRTR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HDFGRTR2_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDFGRTR_EL2, Hypervisor Debug Fine-Grained Read Trap Register

The HDFGRTR_EL2 characteristics are:

Purpose

Provides controls for traps of MRS and MRC reads of debug, trace, PMU, and Statistical Profiling System registers.

Configuration

This register is present only when FEAT_FGT is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HDFGRTR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HDFGRTR_EL2 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56
PMBIDR_EL1	nPMSNEVFR_EL1	nBRBDATA	nBRBCTL	nBRBIDR	PMCEIDn_EL0	PMUSERENR_EL0	TRBTRG_EL1
PMSIRR_EL1	PMSIDR_EL1	PMSICR_EL1	PMSFCR_EL1	PMSEVFR_EL1	PMSCR_EL1	PMBSR_EL1	PMBPTR_EL1
31	30	29	28	27	26	25	24

PMBIDR_EL1, bit [63]

When FEAT_SPE is implemented:

Trap MRS reads of PMBIDR_EL1 at EL1 using AArch64 to EL2.

PMBIDR_EL1	Meaning
0b0	MRS reads of PMBIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMBIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMSNEVFR_EL1, bit [62]

When FEAT_SPE_FnE is implemented:

Trap MRS reads of PMSNEVFR_EL1 at EL1 using AArch64 to EL2.

nPMSNEVFR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSNEVFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of PMSNEVFR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nBRBDATA, bit [61]

When FEAT_BRBE is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- BRBINF<n>_EL1.
- BRBINFINJ_EL1.
- BRBSRC<n>_EL1.
- BRBSRCINJ_EL1.
- BRBTGT<n>_EL1.
- BRBTGTINJ_EL1.
- BRBTS_EL1.

nBRBDATA	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nBRBCTL, bit [60]

When FEAT_BRBE is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- BRBCR_EL1.
- BRBFCR_EL1.

nBRBCTL	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nBRBIDR, bit [59]
When FEAT_BRBE is implemented:

Trap MRS reads of BRBIDR0_EL1 at EL1 using AArch64 to EL2.

nBRBIDR	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of BRBIDR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of BRBIDR0_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMCEIDn_EL0, bit [58]
When FEAT_PMUv3 is implemented:

Trap MRS reads of PMCEID<n>_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMCEID<n> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

PMCEIDn_EL0	Meaning
0b0	MRS reads of PMCEID<n>_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMCEID<n> at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of PMCEID<n>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of PMCEID<n> at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMUSERENR_EL0, bit [57]

When FEAT_PMUv3 is implemented:

Trap MRS reads of PMUSERENR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMUSERENR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

PMUSERENR_EL0	Meaning
0b0	MRS reads of PMUSERENR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMUSERENR at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of PMUSERENR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of PMUSERENR at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBTRG_EL1, bit [56]

When FEAT_TRBE is implemented:

Trap MRS reads of TRBTRG_EL1 at EL1 using AArch64 to EL2.

TRBTRG_EL1	Meaning
0b0	MRS reads of TRBTRG_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBTRG_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBSR_EL1, bit [55]
When FEAT_TRBE is implemented:

Trap MRS reads of TRBSR_EL1 at EL1 using AArch64 to EL2.

TRBSR_EL1	Meaning
0b0	MRS reads of TRBSR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBPTR_EL1, bit [54]
When FEAT_TRBE is implemented:

Trap MRS reads of TRBPTR_EL1 at EL1 using AArch64 to EL2.

TRBPTR_EL1	Meaning
0b0	MRS reads of TRBPTR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBMAR_EL1, bit [53]
When FEAT_TRBE is implemented:

Trap MRS reads of TRBMAR_EL1 at EL1 using AArch64 to EL2.

TRBMAR_EL1	Meaning
0b0	MRS reads of TRBMAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBMAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBLIMITR_EL1, bit [52]

When FEAT_TRBE is implemented:

Trap MRS reads of TRBLIMITR_EL1 at EL1 using AArch64 to EL2.

TRBLIMITR_EL1	Meaning
0b0	MRS reads of TRBLIMITR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBIDR_EL1, bit [51]

When FEAT_TRBE is implemented:

Trap MRS reads of TRBIDR_EL1 at EL1 using AArch64 to EL2.

TRBIDR_EL1	Meaning
0b0	MRS reads of TRBIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBBASER_EL1, bit [50]

When FEAT_TRBE is implemented:

Trap MRS reads of TRBBASER_EL1 at EL1 using AArch64 to EL2.

TRBBASER_EL1	Meaning
0b0	MRS reads of TRBBASER_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBBASER_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [49]

Reserved, RES0.

TRCVICTLR, bit [48]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MRS reads of TRCVICTLR at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MRS reads of ETM TRCVICTLR at EL1 using AArch64 to EL2.

TRCVICTLR	Meaning
0b0	MRS reads of TRCVICTLR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCVICTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCSTATR, bit [47]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MRS reads of TRCSTATR at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MRS reads of ETM TRCSTATR at EL1 using AArch64 to EL2.

TRCSTATR	Meaning
0b0	MRS reads of TRCSTATR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCSTATR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCSSCSRn, bit [46]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented, TRCSSCSR<n> are implemented, and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MRS reads of TRCSSCSR<n> at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MRS reads of ETM TRCSSCSR<n> at EL1 using AArch64 to EL2.

TRCSSCSRn	Meaning
0b0	MRS reads of TRCSSCSR<n> are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCSSCSR<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

If Single-shot Comparator n is not implementented, a read of TRCSSCSR<n> is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCSEQSTR, bit [45]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented, TRCSEQSTR is implemented, and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MRS reads of TRCSEQSTR at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MRS reads of ETM TRCSEQSTR at EL1 using AArch64 to EL2.

TRCSEQSTR	Meaning
0b0	MRS reads of TRCSEQSTR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCSEQSTR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCPRGCTLR, bit [44]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MRS reads of TRCPRGCTLR at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MRS reads of ETM TRCPRGCTLR at EL1 using AArch64 to EL2.

TRCPRGCTLR	Meaning
0b0	MRS reads of TRCPRGCTLR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCPRGCTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCOSLSR, bit [43]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MRS reads of TRCOSLSR at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MRS reads of ETM TRCOSLSR at EL1 using AArch64 to EL2.

TRCOSLSR	Meaning
0b0	MRS reads of TRCOSLSR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCOSLSR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

TRCIMSPECn, bit [41]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MRS reads of TRCIMSPEC<n> at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MRS reads of ETM TRCIMSPEC<n> at EL1 using AArch64 to EL2.

TRCIMSPECn	Meaning
0b0	MRS reads of TRCIMSPEC<n> are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCIMSPEC<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

TRCIMSPEC<1-7> are optional. If TRCIMSPEC<n> is not implemented, a read of TRCIMSPEC<n> is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCID, bit [40]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- In an Armv9 implementation:
  - TRCDEVARCH.
  - TRCDEVID.
  - All of the TRCIDR<n> registers.
- In an Armv8 implementation:
  - ETM TRCDEVARCH.
  - ETM TRCDEVID.
  - All of the ETM TRCIDR<n> registers.

TRCID	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [39:38]

Reserved, RES0.

TRCCNTVRn, bit [37]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented, TRCCNTVR<n> are implemented, and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MRS reads of TRCCNTVR<n> at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MRS reads of ETM TRCCNTVR<n> at EL1 using AArch64 to EL2.

TRCCNTVRn	Meaning
0b0	MRS reads of TRCCNTVR<n> are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCCNTVR<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

If Counter n is not implemented, a read of TRCCNTVR<n> is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCCLAIM, bit [36]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- In an Armv9 implementation: TRCCLAIMCLR and TRCCLAIMSET.
- In an Armv8 implementation: ETM TRCCLAIMCLR and ETM TRCCLAIMSET.

TRCCLAIM	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCAUXCTLR, bit [35]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MRS reads of TRCAUXCTLR at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MRS reads of ETM TRCAUXCTLR at EL1 using AArch64 to EL2.

TRCAUXCTLR	Meaning
0b0	MRS reads of TRCAUXCTLR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCAUXCTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCAUTHSTATUS, bit [34]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MRS reads of TRCAUTHSTATUS at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MRS reads of ETM TRCAUTHSTATUS at EL1 using AArch64 to EL2.

TRCAUTHSTATUS	Meaning
0b0	MRS reads of TRCAUTHSTATUS are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCAUTHSTATUS at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRC, bit [33]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- In an Armv9 implementation:
  - TRCACATR<n>.
  - TRCACVR<n>.
  - TRCBBCTLR.
  - TRCCCCTLR.
  - TRCCIDCCTLR0.
  - TRCCIDCCTLR1.
  - TRCCIDCVR<n>.
  - TRCCNTCTLR<n>.
  - TRCCNTRLDVR<n>.
  - TRCCONFIGR.
  - TRCEVENTCTL0R.
  - TRCEVENTCTL1R.
  - TRCEXTINSELR<n>.
  - TRCQCTLR.
  - TRCRSCTLR<n>.
  - TRCRSR.
  - TRCSEQEVR<n>.
  - TRCSEQRSTEVR.
  - TRCSSCCR<n>.
  - TRCSSPCICR<n>.
  - TRCSTALLCTLR.
  - TRCSYNCPR.
  - TRCTRACEIDR.
  - TRCTSCTLR.
  - TRCVIIECTLR.
  - TRCVIPCSSCTLR.
  - TRCVISSCTLR.
  - TRCVMIDCCTLR0.
  - TRCVMIDCCTLR1.
  - TRCVMIDCVR<n>.
- In an Armv8 implementation:
  - ETM TRCACATR<n>.
  - ETM TRCACVR<n>.
  - ETM TRCBBCTLR.
  - ETM TRCCCCTLR.
  - ETM TRCCIDCCTLR0.
  - ETM TRCCIDCCTLR1.
  - ETM TRCCIDCVR<n>.
  - ETM TRCCNTCTLR<n>.
  - ETM TRCCNTRLDVR<n>.
  - ETM TRCCONFIGR.
  - ETM TRCEVENTCTL0R.
  - ETM TRCEVENTCTL1R.
  - ETM TRCEXTINSELR.
  - ETM TRCQCTLR.
  - ETM TRCRSCTLR<n>.
  - ETM TRCSEQEVR<n>.

- ETM TRCSEQRSTEVR.
- ETM TRCSSCCR<n>.
- ETM TRCSSPCICR<n>.
- ETM TRCSTALLCTLR.
- ETM TRCSYNCPR.
- ETM TRCTRACEIDR.
- ETM TRCTSCTLR.
- ETM TRCVIIECTLR.
- ETM TRCVIPCSSCTLR.
- ETM TRCVISSCTLR.
- ETM TRCVMIDCCTLR0.
- ETM TRCVMIDCCTLR1.
- ETM TRCVMIDCVR<n>.

TRC	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

A read of an unimplemented register is UNDEFINED.

TRCEXTINSELR<n> and TRCRSR are implemented only if FEAT_ETE is implemented.

TRCEXTINSELR is implemented only if FEAT_ETE is not implemented and FEAT_ETMv4 is implemented.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSLATFR_EL1, bit [32]

When FEAT_SPE is implemented:

Trap MRS reads of PMSLATFR_EL1 at EL1 using AArch64 to EL2.

PMSLATFR_EL1	Meaning
0b0	MRS reads of PMSLATFR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSLATFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSIRR_EL1, bit [31]

When FEAT_SPE is implemented:

Trap MRS reads of PMSIRR_EL1 at EL1 using AArch64 to EL2.

PMSIRR_EL1	Meaning
0b0	MRS reads of PMSIRR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSIRR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSIDR_EL1, bit [30]

When FEAT_SPE is implemented:

Trap MRS reads of PMSIDR_EL1 at EL1 using AArch64 to EL2.

PMSIDR_EL1	Meaning
0b0	MRS reads of PMSIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSICR_EL1, bit [29]

When FEAT_SPE is implemented:

Trap MRS reads of PMSICR_EL1 at EL1 using AArch64 to EL2.

PMSICR_EL1	Meaning
0b0	MRS reads of PMSICR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSICR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSFCR_EL1, bit [28]
When FEAT_SPE is implemented:

Trap MRS reads of PMSFCR_EL1 at EL1 using AArch64 to EL2.

PMSFCR_EL1	Meaning
0b0	MRS reads of PMSFCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSFCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSEVFR_EL1, bit [27]
When FEAT_SPE is implemented:

Trap MRS reads of PMSEVFR_EL1 at EL1 using AArch64 to EL2.

PMSEVFR_EL1	Meaning
0b0	MRS reads of PMSEVFR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSEVFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSCR_EL1, bit [26]
When FEAT_SPE is implemented:

Trap MRS reads of PMSCR_EL1 at EL1 using AArch64 to EL2.

PMSCR_EL1	Meaning
0b0	MRS reads of PMSCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMBSR_EL1, bit [25]

When FEAT_SPE is implemented:

Trap MRS reads of PMBSR_EL1 at EL1 using AArch64 to EL2.

PMBSR_EL1	Meaning
0b0	MRS reads of PMBSR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMBPTR_EL1, bit [24]

When FEAT_SPE is implemented:

Trap MRS reads of PMBPTR_EL1 at EL1 using AArch64 to EL2.

PMBPTR_EL1	Meaning
0b0	MRS reads of PMBPTR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMBLIMITR_EL1, bit [23]
When FEAT_SPE is implemented:

Trap MRS reads of PMBLIMITR_EL1 at EL1 using AArch64 to EL2.

PMBLIMITR_EL1	Meaning
0b0	MRS reads of PMBLIMITR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMMIR_EL1, bit [22]
When FEAT_PMUv3 is implemented:

Trap MRS reads of PMMIR_EL1 at EL1 using AArch64 to EL2.

PMMIR_EL1	Meaning
0b0	MRS reads of PMMIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMMIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [21:20]

Reserved, RES0.

PMSELR_EL0, bit [19]

When FEAT_PMUv3 is implemented:

Trap MRS reads of PMSELR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMSELR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

PMSELR_EL0	Meaning
0b0	MRS reads of PMSELR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMSELR at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of PMSELR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of PMSELR at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMOVS, bit [18]

When FEAT_PMUv3 is implemented:

Trap MRS reads and MRC reads of any of the following System registers to EL2:

- At EL1 and EL0 using AArch64: PMOVSCLR_EL0 and PMOVSSET_EL0.
- At EL0 using AArch32 when EL1 is using AArch64: PMOVSR and PMOVSSET.

PMOVS	Meaning
0b0	MRS reads at EL1 and EL0 using AArch64 and MRC reads at EL0 using AArch32 of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads at EL1 and EL0 using AArch64 of any of the specified AArch64 System registers are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads at EL0 using AArch32 when EL1 is using AArch64 of any of the specified AArch32 System registers are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMINTEN, bit [17]

When FEAT_PMUv3 is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- PMINTENCLR_EL1.
- PMINTENSET_EL1.

PMINTEN	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMCNTEN, bit [16]

When FEAT_PMUv3 is implemented:

Trap MRS reads and MRC reads of any of the following System registers to EL2:

- At EL1 and EL0 using AArch64: PMCNTENCLR_EL0 and PMCNTENSET_EL0.
- At EL0 using AArch32 when EL1 is using AArch64: PMCNTENCLR and PMCNTENSET.

PMCNTEN	Meaning
0b0	MRS reads at EL1 and EL0 using AArch64 and MRC reads at EL0 using AArch32 of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads at EL1 and EL0 using AArch64 of any of the specified AArch64 System registers are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads at EL0 using AArch32 when EL1 is using AArch64 of any of the specified AArch32 System registers are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMCCNTR_EL0, bit [15]

When FEAT_PMUv3 is implemented:

Trap MRS reads of PMCCNTR_EL0 at EL1 and EL0 using AArch64 and MRC and MRRC reads of PMCCNTR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

PMCCNTR_EL0	Meaning
0b0	MRS reads of PMCCNTR_EL0 at EL1 and EL0 using AArch64 and MRC and MRRC reads of PMCCNTR at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of PMCCNTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of PMCCNTR at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03. • MRRC reads of PMCCNTR at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x04.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMCCFILTR_EL0, bit [14]

When FEAT_PMUv3 is implemented:

Trap MRS reads of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMCCFILTR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

PMCCFILTR_EL0	Meaning
0b0	MRS reads of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMCCFILTR at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of PMCCFILTR at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

PMCCFILTR_EL0 can also be accessed in AArch64 state using PMXEVTYPER_EL0 when PMSELR_EL0.SEL == 31, and PMCCFILTR can also be accessed in AArch32 state using PMXEVTYPER when PMSELR.SEL == 31.

Setting this field to 1 has no effect on accesses to PMXEVTYPER_EL0 and PMXEVTYPER, regardless of the value of PMSELR_EL0.SEL or PMSELR.SEL.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMEVTYPERn_EL0, bit [13]
When FEAT_PMUv3 is implemented:

Trap MRS reads and MRC reads of any of the following System registers to EL2:

- At EL1 and EL0 using AArch64: PMEVTYPER<n>_EL0 and PMXEVTYPER_EL0.
- At EL0 using AArch32 when EL1 is using AArch64: PMEVTYPER<n> and PMXEVTYPER.

PMEVTYPERn_EL0	Meaning
0b0	MRS reads at EL1 and EL0 using AArch64 and MRC reads at EL0 using AArch32 of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads at EL1 and EL0 using AArch64 of any of the specified AArch64 System registers are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads at EL0 using AArch32 when EL1 is using AArch64 of any of the specified AArch32 System registers are trapped to EL2 and reported with EC syndrome value 0x03.

Regardless of the value of this field, for each value n:

- If event counter n is not implemented, the following accesses are UNDEFINED:
  - In AArch64 state, a read of PMEVTYPER<n>_EL0, or, if n is not 31, a read of PMXEVTYPER_EL0 when PMSELR_EL0.SEL == n.
  - In AArch32 state, a read of PMEVTYPER<n>, or, if n is not 31, a read of PMXEVTYPER when PMSELR.SEL == n.
- If event counter n is implemented, n is greater-than-or-equal-to MDCR_EL2.HPMN, and EL2 is implemented and enabled in the current Security state, the following generate a Trap exception to EL2 from EL0 or EL1:
  - In AArch64 state, a read of PMEVTYPER<n>_EL0, or a read of PMXEVTYPER_EL0 when PMSELR_EL0.SEL == n, reported with EC syndrome value 0x18.
  - In AArch32 state, a read of PMEVTYPER<n>, or a read of PMXEVTYPER when PMSELR.SEL == n, reported with EC syndrome value 0x03.

See also HDFGRTR_EL2.PMCCFILTR_EL0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMEVCNTRn_EL0, bit [12]
When FEAT_PMUv3 is implemented:

Trap MRS reads and MRC reads of any of the following System registers to EL2:

- At EL1 and EL0 using AArch64: PMEVCNTR<n>_EL0 and PMXEVCNTR_EL0.
- At EL0 using AArch32 when EL1 is using AArch64: PMEVCNTR<n> and PMXEVCNTR.

PMEVCNTRn_EL0	Meaning
0b0	MRS reads at EL1 and EL0 using AArch64 and MRC reads at EL0 using AArch32 of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads at EL1 and EL0 using AArch64 of any of the specified AArch64 System registers are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads at EL0 using AArch32 when EL1 is using AArch64 of any of the specified AArch32 System registers are trapped to EL2 and reported with EC syndrome value 0x03.

Regardless of the value of this field, for each value n:

- If event counter n is not implemented, the following accesses are UNDEFINED:
  - In AArch64 state, a read of PMEVCNTR<n>_EL0, or a read of PMXEVCNTR_EL0 when PMSELR_EL0.SEL is n.
  - In AArch32 state, a read of PMEVCNTR<n>, or a read of PMXEVCNTR when PMSELR.SEL is n.
- If event counter n is implemented, n is greater than or equal to MDCR_EL2.HPMN, and EL2 is implemented and enabled in the current Security state, the following generate a Trap exception to EL2 from EL0 or EL1:
  - In AArch64 state, a read of PMEVCNTR<n>_EL0, or a read of PMXEVCNTR_EL0 when PMSELR_EL0.SEL is n, reported with EC syndrome value 0x18.
  - In AArch32 state, a read of PMEVCNTR<n>, or a read of PMXEVCNTR when PMSELR.SEL is n, reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

OSDLR_EL1, bit [11]

When FEAT_DoubleLock is implemented:

Trap MRS reads of OSDLR_EL1 at EL1 using AArch64 to EL2.

OSDLR_EL1	Meaning
0b0	MRS reads of OSDLR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of OSDLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

OSECCR_EL1, bit [10]

Trap MRS reads of OSECCR_EL1 at EL1 using AArch64 to EL2.

OSECCR_EL1	Meaning
0b0	MRS reads of OSECCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of OSECCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

OSLSR_EL1, bit [9]

Trap MRS reads of OSLSR_EL1 at EL1 using AArch64 to EL2.

OSLSR_EL1	Meaning
0b0	MRS reads of OSLSR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of OSLSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

DBGPRCR_EL1, bit [7]

Trap MRS reads of DBGPRCR_EL1 at EL1 using AArch64 to EL2.

DBGPRCR_EL1	Meaning
0b0	MRS reads of DBGPRCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGPRCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DBGAUTHSTATUS_EL1, bit [6]

Trap MRS reads of DBGAUTHSTATUS_EL1 at EL1 using AArch64 to EL2.

DBGAUTHSTATUS_EL1	Meaning
0b0	MRS reads of DBGAUTHSTATUS_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGAUTHSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DBGCLAIM, bit [5]

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- DBGCLAIMCLR_EL1.
- DBGCLAIMSET_EL1.

DBGCLAIM	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

MDSCR_EL1, bit [4]

Trap MRS reads of MDSCR_EL1 at EL1 using AArch64 to EL2.

MDSCR_EL1	Meaning
0b0	MRS reads of MDSCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MDSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DBGWVRn_EL1, bit [3]

Trap MRS reads of DBGWVR<n>_EL1 at EL1 using AArch64 to EL2.

DBGWVRn_EL1	Meaning
0b0	MRS reads of DBGWVR<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGWVR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

If watchpoint n is not implemented, a read of DBGWVR<n>_EL1 is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DBGWCRn_EL1, bit [2]

Trap MRS reads of DBGWCR<n>_EL1 at EL1 using AArch64 to EL2.

DBGWCRn_EL1	Meaning
0b0	MRS reads of DBGWCR<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGWCR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

If watchpoint n is not implemented, a read of DBGWCR<n>_EL1 is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DBGBVRn_EL1, bit [1]

Trap MRS reads of DBGBVR<n>_EL1 at EL1 using AArch64 to EL2.

DBGBVRn_EL1	Meaning
0b0	MRS reads of DBGBVR<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGBVR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

If breakpoint n is not implemented, a read of DBGBVR<n>_EL1 is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DBGBCRn_EL1, bit [0]

Trap MRS reads of DBGBCR<n>_EL1 at EL1 using AArch64 to EL2.

DBGBCRn_EL1	Meaning
0b0	MRS reads of DBGBCR<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGBCR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

If breakpoint n is not implemented, a read of DBGBCR<n>_EL1 is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HDFGRTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HDFGRTR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1D0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HDFGRTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HDFGRTR_EL2;
```

MSR HDFGRTR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1D0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HDFGRTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HDFGRTR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDFGWTR2_EL2, Hypervisor Debug Fine-Grained Write Trap Register 2

The HDFGWTR2_EL2 characteristics are:

Purpose

Provides controls for traps of MSR and MCR writes of debug, trace, PMU, and Statistical Profiling System registers.

Configuration

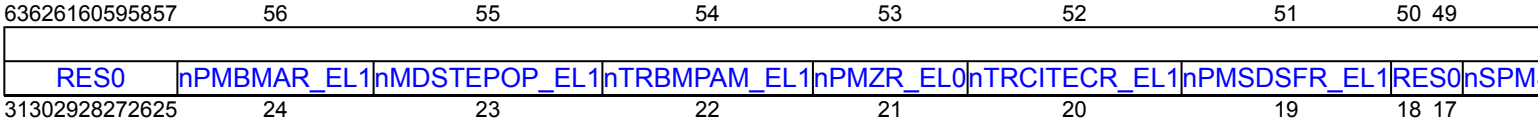
This register is present only when FEAT_FGT2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HDFGWTR2_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HDFGWTR2_EL2 is a 64-bit register.

Field descriptions



Bits [63:25]

Reserved, RES0.

nPMBMAR_EL1, bit [24]

When FEAT_SPE_nVM is implemented:

Trap MSR writes of PMBMAR_EL1 at EL1 using AArch64 to EL2.

nPMBMAR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of PMBMAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PMBMAR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nMDSTEPOP_EL1, bit [23]
When FEAT_STEP2 is implemented:

Trap MSR writes of MDSTEPOP_EL1 at EL1 using AArch64 to EL2.

nMDSTEPOP_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of MDSTEPOP_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of MDSTEPOP_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTRBMPAM_EL1, bit [22]
When FEAT_TRBE_MPAM is implemented:

Trap MSR writes of TRBMPAM_EL1 at EL1 using AArch64 to EL2.

nTRBMPAM_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of TRBMPAM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of TRBMPAM_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMZR_EL0, bit [21]

When FEAT_PMUv3p9 is implemented:

Trap MSR writes of PMZR_EL0 at EL1 and EL0 using AArch64 to EL2.

nPMZR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MSR writes of PMZR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PMZR_EL0 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTRCITECR_EL1, bit [20]

When FEAT_ITE is implemented:

Trap MSR writes of TRCITECR_EL1 at EL1 using AArch64 to EL2.

nTRCITECR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of TRCITECR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of TRCITECR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMSDSFR_EL1, bit [19]

When FEAT_SPE_FDS is implemented:

Trap MSR writes of PMSDSFR_EL1 at EL1 using AArch64 to EL2.

nPMSDSFR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of PMSDSFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PMSDSFR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [18:17]

Reserved, RES0.

nSPMSCR_EL1, bit [16]

When FEAT_SPMU is implemented:

Trap MSR writes of SPMSCR_EL1 at EL1 using AArch64 to EL2.

nSPMSCR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of SPMSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SPMSCR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMACCESSR_EL1, bit [15]

When FEAT_SPMU is implemented:

Trap MSR writes of SPMACCESSR_EL1 at EL1 using AArch64 to EL2.

nSPMACCESSR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of SPMACCESSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SPMACCESSR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMCR_EL0, bit [14]

When FEAT_SPMU is implemented:

Trap MSR writes of SPMCR_EL0 at EL1 and EL0 using AArch64 to EL2.

nSPMCR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MSR writes of SPMCR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SPMCR_EL0 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMOVS, bit [13]

When FEAT_SPMU is implemented:

Trap MSR writes at EL1 and EL0 using AArch64 of any of the following AArch64 System registers to EL2:

- SPMOVSCLR_EL0.
- SPMOVSSET_EL0.

nSPMOVS	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MSR writes at EL1 and EL0 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMINTEN, bit [12]

When FEAT_SPMU is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- SPMINTENCLR_EL1.
- SPMINTENSET_EL1.

nSPMINTEN	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMCNTEN, bit [11]

When FEAT_SPMU is implemented:

Trap MSR writes at EL1 and EL0 using AArch64 of any of the following AArch64 System registers to EL2:

- SPMCNTENCLR_EL0.
- SPMCNTENSET_EL0.

nSPMCNTEN	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MSR writes at EL1 and EL0 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMSELR_EL0, bit [10]

When FEAT_SPMU is implemented:

Trap MSR writes of SPMSELR_EL0 at EL1 and EL0 using AArch64 to EL2.

nSPMSELR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MSR writes of SPMSELR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SPMSELR_EL0 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMEVTYPERn_EL0, bit [9]

When FEAT_SPMU is implemented:

Trap MSR writes at EL1 and EL0 using AArch64 of any of the following AArch64 System registers to EL2:

- SPMEVTYPER<n>_EL0.
- SPMEVFILTR<n>_EL0.
- SPMEVFILT2R<n>_EL0.

nSPMEVTYPERn_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MSR writes at EL1 and EL0 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSPMEVCNTRn_EL0, bit [8]
When FEAT_SPMU is implemented:

Trap MSR writes at EL1 and EL0 using AArch64 of any of the following AArch64 System registers to EL2:

- SPMEVCNTR<n>_EL0.
- SPMZR_EL0.

nSPMEVCNTRn_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then MSR writes at EL1 and EL0 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMSSCR_EL1, bit [7]
When FEAT_PMUv3_SS is implemented:

Trap MSR writes of PMSSCR_EL1 at EL1 using AArch64 to EL2.

nPMSSCR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of PMSSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PMSSCR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [6]

Reserved, RES0.

nMDSELR_EL1, bit [5]

When FEAT_Debugv8p9 is implemented:

Trap MSR writes of MDSELR_EL1 at EL1 using AArch64 to EL2.

nMDSELR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of MDSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of MDSELR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMUACR_EL1, bit [4]

When FEAT_PMUv3p9 is implemented:

Trap MSR writes of PMUACR_EL1 at EL1 using AArch64 to EL2.

nPMUACR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of PMUACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PMUACR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMICFILTR_EL0, bit [3]

When FEAT_PMUv3_ICNTR is implemented:

Trap MSR writes of PMICFILTR_EL0 at EL1 and EL0 using AArch64 to EL2.

nPMICFILTR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then: • MSR writes of PMICFILTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. • PMCNTENCLR_EL0.F0, PMCNTENSET_EL0.F0, PMOVSCLR_EL0.F0, and PMOVSSET_EL0.F0 ignore writes at EL1 and EL0. • PMINTENCLR_EL1.F0 and PMINTENSET_EL1.F0 ignore writes at EL1.
0b1	MSR writes of PMICFILTR_EL0 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMICNTR_EL0, bit [2]

When FEAT_PMUv3_ICNTR is implemented:

Trap MSR writes of PMICNTR_EL0 at EL1 and EL0 using AArch64 to EL2.

nPMICNTR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then: • MSR writes of PMICNTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. • PMZR_EL0.F0 ignores writes at EL1 and EL0.
0b1	MSR writes of PMICNTR_EL0 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMIAR_EL1, bit [1]
When FEAT_SEBEP is implemented:

Trap MSR writes of PMIAR_EL1 at EL1 using AArch64 to EL2.

nPMIAR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of PMIAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PMIAR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPMECR_EL1, bit [0]
When FEAT_EBEP is implemented or FEAT_PMUv3_SS is implemented:

Trap MSR writes of PMECR_EL1 at EL1 using AArch64 to EL2.

nPMECR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of PMECR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PMECR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing HDFGWTR2_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HDFGWTR2_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1B0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HDFGWTR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HDFGWTR2_EL2;
```

MSR HDFGWTR2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1B0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HDFGWTR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HDFGWTR2_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDFGWTR_EL2, Hypervisor Debug Fine-Grained Write Trap Register

The HDFGWTR_EL2 characteristics are:

Purpose

Provides controls for traps of MSR and MCR writes of debug, trace, PMU, and Statistical Profiling System registers.

Configuration

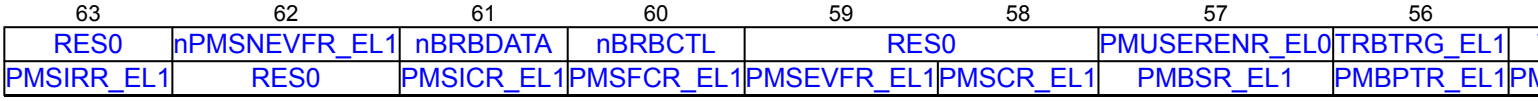
This register is present only when FEAT_FGT is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HDFGWTR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HDFGWTR_EL2 is a 64-bit register.

Field descriptions



Bit [63]

Reserved, RES0.

nPMSNEVFR_EL1, bit [62]

When FEAT_SPE_FnE is implemented:

Trap MSR writes of PMSNEVFR_EL1 at EL1 using AArch64 to EL2.

nPMSNEVFR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSNEVFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PMSNEVFR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nBRBDATA, bit [61]

When FEAT_BRBE is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- BRBINFINJ_EL1.
- BRBSRCINJ_EL1.
- BRBTGTINJ_EL1.
- BRBTS_EL1.

nBRBDATA	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nBRBCTL, bit [60]

When FEAT_BRBE is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- BRBCR_EL1.
- BRBFCR_EL1.

nBRBCTL	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [59:58]

Reserved, RES0.

PMUSERENR_EL0, bit [57]

When FEAT_PMUv3 is implemented:

Trap MSR writes of PMUSERENR_EL0 at EL1 using AArch64 to EL2.

PMUSERENR_EL0	Meaning
0b0	MSR writes of PMUSERENR_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMUSERENR_EL0 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBTRG_EL1, bit [56]

When FEAT_TRBE is implemented:

Trap MSR writes of TRBTRG_EL1 at EL1 using AArch64 to EL2.

TRBTRG_EL1	Meaning
0b0	MSR writes of TRBTRG_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBTRG_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBSR_EL1, bit [55]

When FEAT_TRBE is implemented:

Trap MSR writes of TRBSR_EL1 at EL1 using AArch64 to EL2.

TRBSR_EL1	Meaning
0b0	MSR writes of TRBSR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBPTR_EL1, bit [54]
When FEAT_TRBE is implemented:

Trap MSR writes of TRBPTR_EL1 at EL1 using AArch64 to EL2.

TRBPTR_EL1	Meaning
0b0	MSR writes of TRBPTR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBMAR_EL1, bit [53]
When FEAT_TRBE is implemented:

Trap MSR writes of TRBMAR_EL1 at EL1 using AArch64 to EL2.

TRBMAR_EL1	Meaning
0b0	MSR writes of TRBMAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBMAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBLIMITR_EL1, bit [52]
When FEAT_TRBE is implemented:

Trap MSR writes of TRBLIMITR_EL1 at EL1 using AArch64 to EL2.

TRBLIMITR_EL1	Meaning
0b0	MSR writes of TRBLIMITR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [51]

Reserved, RES0.

TRBBASER_EL1, bit [50]

When FEAT_TRBE is implemented:

Trap MSR writes of TRBBASER_EL1 at EL1 using AArch64 to EL2.

TRBBASER_EL1	Meaning
0b0	MSR writes of TRBBASER_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBBASER_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRFCR_EL1, bit [49]

When FEAT_TRF is implemented:

Trap MSR writes of TRFCR_EL1 at EL1 using AArch64 to EL2.

TRFCR_EL1	Meaning
0b0	MSR writes of TRFCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRFCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCVICTLR, bit [48]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MSR writes of TRCVICTLR at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MSR writes of ETM TRCVICTLR at EL1 using AArch64 to EL2.

TRCVICTLR	Meaning
0b0	MSR writes of TRCVICTLR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCVICTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [47]

Reserved, RES0.

TRCSSCSRn, bit [46]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented, TRCSSCSR<n> are implemented, and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MSR writes of TRCSSCSR<n> at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MSR writes of ETM TRCSSCSR<n> at EL1 using AArch64 to EL2.

TRCSSCSRn	Meaning
0b0	MSR writes of TRCSSCSR<n> are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCSSCSR<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

If Single-shot Comparator n is not implementented, a write of TRCSSCSR<n> is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCSEQSTR, bit [45]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented, TRCSEQSTR is implemented, and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MSR writes of TRCSEQSTR at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MSR writes of ETM TRCSEQSTR at EL1 using AArch64 to EL2.

TRCSEQSTR	Meaning
0b0	MSR writes of TRCSEQSTR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCSEQSTR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCPRGCTLR, bit [44]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MSR writes of TRCPRGCTLR at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MSR writes of ETM TRCPRGCTLR at EL1 using AArch64 to EL2.

TRCPRGCTLR	Meaning
0b0	MSR writes of TRCPRGCTLR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCPRGCTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [43]

Reserved, RES0.

TRCOSLAR, bit [42]

When FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented:

In an Armv8 implementation, trap MSR writes of ETM TRCOSLAR at EL1 using AArch64 to EL2.

TRCOSLAR	Meaning
0b0	MSR writes of ETM TRCOSLAR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ETM TRCOSLAR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCIMSPECn, bit [41]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MSR writes of TRCIMSPEC<n> at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MSR writes of ETM TRCIMSPEC<n> at EL1 using AArch64 to EL2.

TRCIMSPECn	Meaning
0b0	MSR writes of TRCIMSPEC<n> are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCIMSPEC<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

TRCIMSPEC<1-7> are optional. If TRCIMSPEC<n> is not implemented, a write of TRCIMSPEC<n> is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [40:38]

Reserved, RES0.

TRCCNTVRn, bit [37]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented, TRCCNTVR<n> are implemented, and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MSR writes of TRCCNTVR<n> at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MSR writes of ETM TRCCNTVR<n> at EL1 using AArch64 to EL2.

TRCCNTVRn	Meaning
0b0	MSR writes of TRCCNTVR<n> are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCCNTVR<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

If Counter n is not implemented, a write of TRCCNTVR<n> is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCCLAIM, bit [36]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- In an Armv9 implementation: TRCCLAIMCLR and TRCCLAIMSET.
- In an Armv8 implementation: ETM TRCCLAIMCLR and ETM TRCCLAIMSET.

TRCCLAIM	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCAUXCTLR, bit [35]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

In an Armv9 implementation, trap MSR writes of TRCAUXCTLR at EL1 using AArch64 to EL2.

In an Armv8 implementation, trap MSR writes of ETM TRCAUXCTLR at EL1 using AArch64 to EL2.

TRCAUXCTLR	Meaning
0b0	MSR writes of TRCAUXCTLR are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCAUXCTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [34]

Reserved, RES0.

TRC, bit [33]

When FEAT_ETE is implemented or (FEAT_ETMv4 is implemented and System register access to the trace unit registers is implemented):

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- In an Armv9 implementation:
  - TRCACATR<n>.
  - TRCACVR<n>.
  - TRCBBCTLR.
  - TRCCCCTLR.
  - TRCCIDCCTLR0.
  - TRCCIDCCTLR1.
  - TRCCIDCVR<n>.
  - TRCCNTCTLR<n>.
  - TRCCNTRLDVR<n>.
  - TRCCONFIGR.
  - TRCEVENTCTL0R.
  - TRCEVENTCTL1R.
  - TRCEXTINSELR<n>.
  - TRCQCTLR.
  - TRCRSCTLR<n>.
  - TRCRSR.
  - TRCSEQEVR<n>.
  - TRCSEQRSTEVR.
  - TRCSSCCR<n>.
  - TRCSSPCICR<n>.
  - TRCSTALLCTLR.
  - TRCSYNCPR.
  - TRCTRACEIDR.
  - TRCTSCTLR.
  - TRCVIIECTLR.
  - TRCVIPCSSCTLR.
  - TRCVISSCTLR.
  - TRCVMIDCCTLR0.
  - TRCVMIDCCTLR1.
  - TRCVMIDCVR<n>.
- In an Armv8 implementation:
  - ETM TRCACATR<n>.
  - ETM TRCACVR<n>.

- ETM TRCBBCTLR.
- ETM TRCCCCTLR.
- ETM TRCCIDCCTLR0.
- ETM TRCCIDCCTLR1.
- ETM TRCCIDCVR<n>.
- ETM TRCCNTCTLR<n>.
- ETM TRCCNTRLDVR<n>.
- ETM TRCCONFIGR.
- ETM TRCEVENTCTL0R.
- ETM TRCEVENTCTL1R.
- ETM TRCEXTINSELR.
- ETM TRCQCTLR.
- ETM TRCRSCTLR<n>.
- ETM TRCSEQEVR<n>.
- ETM TRCSEQRSTEVR.
- ETM TRCSSCCR<n>.
- ETM TRCSSPCICR<n>.
- ETM TRCSTALLCTLR.
- ETM TRCSYNCPR.
- ETM TRCTRACEIDR.
- ETM TRCTSCTLR.
- ETM TRCVIIECTLR.
- ETM TRCVIPCSSCTLR.
- ETM TRCVISSCTLR.
- ETM TRCVMIDCCTLR0.
- ETM TRCVMIDCCTLR1.
- ETM TRCVMIDCVR<n>.

TRC	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

A write of an unimplemented register is UNDEFINED.

TRCEXTINSELR<n> and TRCRSR are implemented only if FEAT_ETE is implemented.

TRCEXTINSELR is implemented only if FEAT_ETE is not implemented and FEAT_ETMv4 is implemented.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSLATFR_EL1, bit [32]
When FEAT_SPE is implemented:

Trap MSR writes of PMSLATFR_EL1 at EL1 using AArch64 to EL2.

PMSLATFR_EL1	Meaning
0b0	MSR writes of PMSLATFR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSLATFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSIRR_EL1, bit [31]

When FEAT_SPE is implemented:

Trap `MSR` writes of PMSIRR_EL1 at EL1 using AArch64 to EL2.

PMSIRR_EL1	Meaning
`0b0`	`MSR` writes of PMSIRR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then `MSR` writes of PMSIRR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [30]

Reserved, RES0.

PMSICR_EL1, bit [29]

When FEAT_SPE is implemented:

Trap `MSR` writes of PMSICR_EL1 at EL1 using AArch64 to EL2.

PMSICR_EL1	Meaning
`0b0`	`MSR` writes of PMSICR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then `MSR` writes of PMSICR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSFCR_EL1, bit [28]

When FEAT_SPE is implemented:

Trap MSR writes of PMSFCR_EL1 at EL1 using AArch64 to EL2.

PMSFCR_EL1	Meaning
0b0	MSR writes of PMSFCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSFCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSEVFR_EL1, bit [27]

When FEAT_SPE is implemented:

Trap MSR writes of PMSEVFR_EL1 at EL1 using AArch64 to EL2.

PMSEVFR_EL1	Meaning
0b0	MSR writes of PMSEVFR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSEVFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSCR_EL1, bit [26]

When FEAT_SPE is implemented:

Trap MSR writes of PMSCR_EL1 at EL1 using AArch64 to EL2.

PMSCR_EL1	Meaning
0b0	MSR writes of PMSCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMBSR_EL1, bit [25]

When FEAT_SPE is implemented:

Trap MSR writes of PMBSR_EL1 at EL1 using AArch64 to EL2.

PMBSR_EL1	Meaning
0b0	MSR writes of PMBSR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMBPTR_EL1, bit [24]

When FEAT_SPE is implemented:

Trap MSR writes of PMBPTR_EL1 at EL1 using AArch64 to EL2.

PMBPTR_EL1	Meaning
0b0	MSR writes of PMBPTR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMBLIMITR_EL1, bit [23]

When FEAT_SPE is implemented:

Trap MSR writes of PMBLIMITR_EL1 at EL1 using AArch64 to EL2.

PMBLIMITR_EL1	Meaning
0b0	MSR writes of PMBLIMITR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [22]

Reserved, RES0.

PMCR_EL0, bit [21]

When FEAT_PMUv3 is implemented:

Trap MSR writes of PMCR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMCR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

PMCR_EL0	Meaning
0b0	MSR writes of PMCR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMCR at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of PMCR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes of PMCR at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSWINC_EL0, bit [20]

When FEAT_PMUv3 is implemented:

Trap MSR writes of PMSWINC_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMSWINC at EL0 using AArch32 when EL1 is using AArch64 to EL2.

PMSWINC_EL0	Meaning
0b0	MSR writes of PMSWINC_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMSWINC at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of PMSWINC_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes of PMSWINC at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSELR_EL0, bit [19]
When FEAT_PMUv3 is implemented:

Trap MSR writes of PMSELR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMSELR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

PMSELR_EL0	Meaning
0b0	MSR writes of PMSELR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMSELR at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of PMSELR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes of PMSELR at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMOVS, bit [18]
When FEAT_PMUv3 is implemented:

Trap MSR writes and MCR writes of any of the following System registers to EL2:

- At EL1 and EL0 using AArch64: PMOVSCLR_EL0 and PMOVSSET_EL0.
- At EL0 using AArch32 when EL1 is using AArch64: PMOVSR and PMOVSSET.

PMOVS	Meaning
0b0	MSR writes at EL1 and EL0 using AArch64 and MCR writes at EL0 using AArch32 of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes at EL1 and EL0 using AArch64 of any of the specified AArch64 System registers are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes at EL0 using AArch32 when EL1 is using AArch64 of any of the specified AArch32 System registers are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMINTEN, bit [17]
When FEAT_PMUv3 is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- PMINTENCLR_EL1.
- PMINTENSET_EL1.

PMINTEN	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMCNTEN, bit [16]
When FEAT_PMUv3 is implemented:

Trap MSR writes and MCR writes of any of the following System registers to EL2:

- At EL1 and EL0 using AArch64: PMCNTENCLR_EL0 and PMCNTENSET_EL0.
- At EL0 using AArch32 when EL1 is using AArch64: PMCNTENCLR and PMCNTENSET.

PMCNTEN	Meaning
0b0	MSR writes at EL1 and EL0 using AArch64 and MCR writes at EL0 using AArch32 of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes at EL1 and EL0 using AArch64 of any of the specified AArch64 System registers are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes at EL0 using AArch32 when EL1 is using AArch64 of any of the specified AArch32 System registers are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMCCNTR_EL0, bit [15]

When FEAT_PMUv3 is implemented:

Trap MSR writes of PMCCNTR_EL0 at EL1 and EL0 using AArch64 and MCR and MCRR writes of PMCCNTR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

PMCCNTR_EL0	Meaning
0b0	MSR writes of PMCCNTR_EL0 at EL1 and EL0 using AArch64 and MCR and MCRR writes of PMCCNTR at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of PMCCNTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes of PMCCNTR at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03. • MCRR writes of PMCCNTR at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x04.

PMCCNTR_EL0 can also be indirectly set to zero by a write of 1 to PMCR_EL0.C or PMZR_EL0.C in AArch64 state, or a write of 1 to PMCR.C in AArch32 state. Setting this field to 1 has no effect on indirect writes to PMCCNTR_EL0 using PMCR_EL0, PMZR_EL0, or PMCR.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMCCFILTR_EL0, bit [14]

When FEAT_PMUv3 is implemented:

Trap MSR writes of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMCCFILTR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

PMCCFILTR_EL0	Meaning
0b0	MSR writes of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMCCFILTR at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes of PMCCFILTR at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

PMCCFILTR_EL0 can also be accessed in AArch64 state using PMXEVTYPER_EL0 when PMSELR_EL0.SEL == 31, and PMCCFILTR can also be accessed in AArch32 state using PMXEVTYPER when PMSELR.SEL == 31.

Setting this field to 1 has no effect on accesses to PMXEVTYPER_EL0 and PMXEVTYPER, regardless of the value of PMSELR_EL0.SEL or PMSELR.SEL.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMEVTYPERn_EL0, bit [13]

When FEAT_PMUv3 is implemented:

Trap MSR writes and MCR writes of any of the following System registers to EL2:

- At EL1 and EL0 using AArch64: PMEVTYPER<n>_EL0 and PMXEVTYPER_EL0.
- At EL0 using AArch32 when EL1 is using AArch64: PMEVTYPER<n> and PMXEVTYPER.

PMEVTYPERn_EL0	Meaning
0b0	MSR writes at EL1 and EL0 using AArch64 and MCR writes at EL0 using AArch32 of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes at EL1 and EL0 using AArch64 of any of the specified AArch64 System registers are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes at EL0 using AArch32 when EL1 is using AArch64 of any of the specified AArch32 System registers are trapped to EL2 and reported with EC syndrome value 0x03.

Regardless of the value of this field, for each value n:

- If event counter n is not implemented, the following accesses are UNDEFINED:
  - In AArch64 state, a write of PMEVTYPER<n>_EL0, or, if n is not 31, a write of PMXEVTYPER_EL0 when PMSELR_EL0.SEL == n.

        - In AArch32 state, a write of PMEVTYPER<n>, or, if n is not 31, a write of PMXEVTYPER when PMSELR.SEL == n.
    - If event counter n is implemented, n is greater-than-or-equal-to MDCR_EL2.HPMN, and EL2 is implemented and enabled in the current Security state, the following generate a Trap exception to EL2 from EL0 or EL1:
        - In AArch64 state, a write of PMEVTYPER<n>_EL0, or a write of PMXEVTYPER_EL0 when PMSELR_EL0.SEL == n, reported with EC syndrome value 0x18.
        - In AArch32 state, a write of PMEVTYPER<n>, or a write of PMXEVTYPER when PMSELR.SEL == n, reported with EC syndrome value 0x03.

See also HDFGWTR_EL2.PMCCFILTR_EL0.

The reset behavior of this field is:

- On a Warm reset:
    - When the highest implemented Exception level is EL2, this field resets to '0'.
    - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMEVCNTRn_EL0, bit [12]
When FEAT_PMUv3 is implemented:

Trap MSR writes and MCR writes of any of the following System registers to EL2:

- At EL1 and EL0 using AArch64: PMEVCNTR<n>_EL0 and PMXEVCNTR_EL0.
- At EL0 using AArch32 when EL1 is using AArch64: PMEVCNTR<n> and PMXEVCNTR.

PMEVCNTRn_EL0	Meaning
0b0	MSR writes at EL1 and EL0 using AArch64 and MCR writes at EL0 using AArch32 of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes at EL1 and EL0 using AArch64 of any of the specified AArch64 System registers are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes at EL0 using AArch32 when EL1 is using AArch64 of any of the specified AArch32 System registers are trapped to EL2 and reported with EC syndrome value 0x03.

Regardless of the value of this field, for each value n:

- If event counter n is not implemented, the following accesses are UNDEFINED:
    - In AArch64 state, a write of PMEVCNTR<n>_EL0, or a write of PMXEVCNTR_EL0 when PMSELR_EL0.SEL is n.
    - In AArch32 state, a write of PMEVCNTR<n>, or a write of PMXEVCNTR when PMSELR.SEL is n.
- If event counter n is implemented, n is greater than or equal to MDCR_EL2.HPMN, and EL2 is implemented and enabled in the current Security state, the following generate a Trap exception to EL2 from EL0 or EL1:
    - In AArch64 state, a write of PMEVCNTR<n>_EL0, or a write of PMXEVCNTR_EL0 when PMSELR_EL0.SEL is n, reported with EC syndrome value 0x18.
    - In AArch32 state, a write of PMEVCNTR<n>, or a write of PMXEVCNTR when PMSELR.SEL is n, reported with EC syndrome value 0x03.

For values of n less than MDCR_EL2.HPMN, PMEVCNTR<n>_EL0 can also be indirectly set to zero by a write of 1 to PMCR_EL0.P or PMZR_EL0.P<n> in AArch64 state, or a write of 1 to PMCR.P in AArch32 state. Setting this field to 1 has no effect on indirect writes to PMEVCNTR<n>_EL0 using PMCR_EL0, PMZR_EL0, or PMCR.

The reset behavior of this field is:

- On a Warm reset:
    - When the highest implemented Exception level is EL2, this field resets to '0'.
    - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

OSDLR_EL1, bit [11]

When FEAT_DoubleLock is implemented:

Trap MSR writes of OSDLR_EL1 at EL1 using AArch64 to EL2.

OSDLR_EL1	Meaning
0b0	MSR writes of OSDLR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of OSDLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

OSECCR_EL1, bit [10]

Trap MSR writes of OSECCR_EL1 at EL1 using AArch64 to EL2.

OSECCR_EL1	Meaning
0b0	MSR writes of OSECCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of OSECCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

OSLAR_EL1, bit [8]

Trap MSR writes of OSLAR_EL1 at EL1 using AArch64 to EL2.

OSLAR_EL1	Meaning
0b0	MSR writes of OSLAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of OSLAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DBGPRCR_EL1, bit [7]

Trap MSR writes of DBGPRCR_EL1 at EL1 using AArch64 to EL2.

DBGPRCR_EL1	Meaning
0b0	MSR writes of DBGPRCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of DBGPRCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

DBGCLAIM, bit [5]

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- DBGCLAIMCLR_EL1.
- DBGCLAIMSET_EL1.

DBGCLAIM	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

MDSCR_EL1, bit [4]

Trap MSR writes of MDSCR_EL1 at EL1 using AArch64 to EL2.

MDSCR_EL1	Meaning
0b0	MSR writes of MDSCR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of MDSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

- Otherwise, this field resets to an architecturally UNKNOWN value.

DBGWVRn_EL1, bit [3]

Trap MSR writes of DBGWVR<n>_EL1 at EL1 using AArch64 to EL2.

DBGWVRn_EL1	Meaning
0b0	MSR writes of DBGWVR<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of DBGWVR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

If watchpoint n is not implemented, a write of DBGWVR<n>_EL1 is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DBGWCRn_EL1, bit [2]

Trap MSR writes of DBGWCR<n>_EL1 at EL1 using AArch64 to EL2.

DBGWCRn_EL1	Meaning
0b0	MSR writes of DBGWCR<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of DBGWCR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

If watchpoint n is not implemented, a write of DBGWCR<n>_EL1 is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DBGBVRn_EL1, bit [1]

Trap MSR writes of DBGBVR<n>_EL1 at EL1 using AArch64 to EL2.

DBGBVRn_EL1	Meaning
0b0	MSR writes of DBGBVR<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of DBGBVR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

If breakpoint n is not implemented, a write of DBGBVR<n>_EL1 is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DBGBCRn_EL1, bit [0]

Trap MSR writes of DBGBCR<n>_EL1 at EL1 using AArch64 to EL2.

DBGBCRn_EL1	Meaning
0b0	MSR writes of DBGBCR<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of DBGBCR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

If breakpoint n is not implemented, a write of DBGBCR<n>_EL1 is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HDFGWTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HDFGWTR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1D8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HDFGWTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HDFGWTR_EL2;
```

```
MSR HDFGWTR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1D8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HDFGWTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HDFGWTR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HFGITR2_EL2, Hypervisor Fine-Grained Instruction Trap Register 2

The HFGITR2_EL2 characteristics are:

Purpose

Provides instruction trap controls.

Configuration

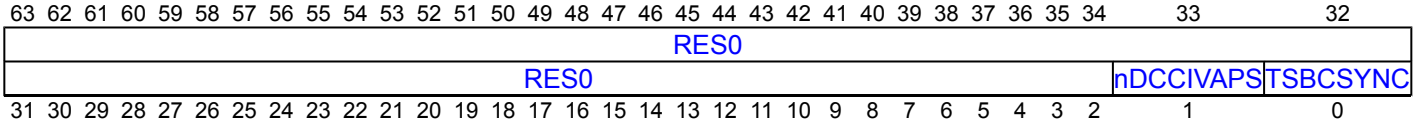
This register is present only when FEAT_FGT2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HFGITR2_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HFGITR2_EL2 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

nDCCIVAPS, bit [1]

When FEAT_PoPS is implemented:

Trap execution of any of the following AArch64 instructions at EL1 to EL2:

- DC CIVAPS.
- DC CIGDVAPS, if FEAT_MTE2 is implemented.

If the Point of Physical Storage is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of the affected instruction is trapped when the value of this control is 1.

nDCCIVAPS	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then execution at EL1 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.
0b1	Execution of the specified instructions is not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSBCSYNC, bit [0]
When FEAT_TRBEv1p1 is implemented:

Trap execution of TSB CSYNC at EL1 and EL0 using AArch64 to EL2.

TSBCSYNC	Meaning
0b0	Execution of TSB CSYNC is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then execution of TSB CSYNC at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x0A, unless the instruction generates a higher priority exception.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing HFGITR2_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HFGITR2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b111

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x310];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HFGITR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HFGITR2_EL2;
```

```
MSR HFGITR2_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b111

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x310] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGITR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HFGITR2_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HFGITR_EL2, Hypervisor Fine-Grained Instruction Trap Register

The HFGITR_EL2 characteristics are:

Purpose

Provides instruction trap controls.

Configuration

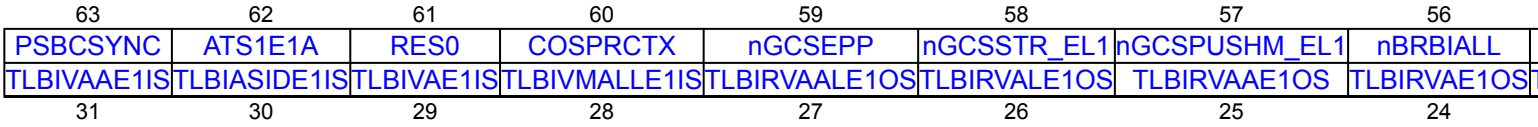
This register is present only when FEAT_FGT is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HFGITR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HFGITR_EL2 is a 64-bit register.

Field descriptions



PSBCSYNC, bit [63]

When FEAT_SPEv1p5 is implemented:

Trap execution of PSB CSYNC at EL1 and EL0 using AArch64 to EL2.

PSBCSYNC	Meaning
0b0	Execution of PSB CSYNC is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of PSB CSYNC at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x0A, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATS1E1A, bit [62]

When FEAT_ATS1A is implemented:

Trap execution of AT S1E1A at EL1 using AArch64 to EL2.

ATS1E1A	Meaning
0b0	Execution of AT S1E1A is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E1A at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [61]

Reserved, RES0.

COSPRCTX, bit [60]

When FEAT_SPECRES2 is implemented:

Trap execution of COSP RCTX at EL1 and EL0 using AArch64 and execution of COSPRCTX at EL0 using AArch32 when EL1 is using AArch64 to EL2.

COSPRCTX	Meaning
0b0	Execution of COSP RCTX at EL1 and EL0 using AArch64 and execution of COSPRCTX at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the instruction generates a higher priority exception: • Execution of COSP RCTX at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18. • Execution of COSPRCTX at EL0 using AArch32 when EL1 is using AArch64 is trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCSEPP, bit [59]

When FEAT_GCS is implemented:

Trap execution of any of the following AArch64 instructions at EL1 to EL2:

- GCSPUSHX.
- GCSPOPCX.

nGCSEPP	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution at EL1 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.
0b1	Execution of the specified instructions is not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCSSTR_EL1, bit [58]

When FEAT_GCS is implemented:

Trap execution of any of the following AArch64 instructions at EL1 to EL2:

- GCSSTR.
- GCSSTTR when PSTATE.UAO is 1.
- GCSSTTR when the Effective value of HCR_EL2.{NV, NV1} is {1, 1}.

nGCSSTR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution at EL1 using AArch64 of any of the specified instructions generates a GCS exception with EC syndrome value 0x2D, unless the instruction generates a higher priority exception.
0b1	Execution of the specified instructions is not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCSPUSHM_EL1, bit [57]

When FEAT_GCS is implemented:

Trap execution of GCSPUSHM at EL1 using AArch64 to EL2.

nGCSPUSHM_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of GCSPUSHM at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.
0b1	Execution of GCSPUSHM is not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nBRBIALL, bit [56]

When FEAT_BRBE is implemented:

Trap execution of BRB IALL at EL1 using AArch64 to EL2.

nBRBIALL	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of BRB IALL at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.
0b1	Execution of BRB IALL is not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nBRBINJ, bit [55]

When FEAT_BRBE is implemented:

Trap execution of BRB INJ at EL1 using AArch64 to EL2.

nBRBINJ	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of BRB INJ at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.
0b1	Execution of BRB INJ is not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DCCVAC, bit [54]

Trap execution of any of the following AArch64 instructions at EL1 and EL0 to EL2:

- DC CVAC.
- DC CGVAC, if FEAT_MTE is implemented.
- DC CGDVAC, if FEAT_MTE is implemented.

- DC CVAOC, if FEAT_OCCMO is implemented.
- DC CGDVAOC, if FEAT_OCCMO is implemented.

DCCVAC	Meaning
0b0	Execution of the specified instructions is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then if the execution can be trapped, execution at EL1 and EL0 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SVC_EL1, bit [53]

Trap execution of SVC at EL1 using AArch64 to EL2.

SVC_EL1	Meaning
0b0	Execution of SVC is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of SVC at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x15, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SVC_EL0, bit [52]

Trap execution of SVC at EL0 using AArch64 and execution of SVC at EL0 using AArch32 when EL1 is using AArch64 to EL2.

SVC_EL0	Meaning
0b0	Execution of SVC at EL0 using AArch64 and execution of SVC at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the instruction generates a higher priority exception: • Execution of SVC at EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x15. • Execution of SVC at EL0 using AArch32 when EL1 is using AArch64 is trapped to EL2 and reported with EC syndrome value 0x11.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

ERET, bit [51]

Trap execution of any of the following AArch64 instructions at EL1 to EL2:

- ERET.
- ERETAA, if FEAT_PAuth is implemented.
- ERETAB, if FEAT_PAuth is implemented.

ERET	Meaning
0b0	Execution of the specified instructions is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution at EL1 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x1A, unless the instruction generates a higher priority exception.

If EL2 is implemented and enabled in the current Security state, HCR_EL2.API == 0, and this field enables a fine-grained trap on the instruction, then execution at EL1 using AArch64 of ERETAA or ERETAB instructions is trapped to EL2 and reported with EC syndrome value 0x1A with its associated ISS field, as the fine-grained trap has higher priority than the trap enabled by HCR_EL2.API == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CPPRCTX, bit [50]

When FEAT_SPECRES is implemented:

Trap execution of CPP RCTX at EL1 and EL0 using AArch64 and execution of CPPRCTX at EL0 using AArch32 when EL1 is using AArch64 to EL2.

CPPRCTX	Meaning
0b0	Execution of CPP RCTX at EL1 and EL0 using AArch64 and execution of CPPRCTX at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the instruction generates a higher priority exception: • Execution of CPP RCTX at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18. • Execution of CPPRCTX at EL0 using AArch32 when EL1 is using AArch64 is trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DVPRCTX, bit [49]

When FEAT_SPECRES is implemented:

Trap execution of DVP RCTX at EL1 and EL0 using AArch64 and execution of DVPRCTX at EL0 using AArch32 when EL1 is using AArch64 to EL2.

DVPRCTX	Meaning
0b0	Execution of DVP RCTX at EL1 and EL0 using AArch64 and execution of DVPRCTX at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the instruction generates a higher priority exception: • Execution of DVP RCTX at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18. • Execution of DVPRCTX at EL0 using AArch32 when EL1 is using AArch64 is trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CFPRCTX, bit [48]

When FEAT_SPECRES is implemented:

Trap execution of CFP RCTX at EL1 and EL0 using AArch64 and execution of CFPRCTX at EL0 using AArch32 when EL1 is using AArch64 to EL2.

CFPRCTX	Meaning
0b0	Execution of CFP RCTX at EL1 and EL0 using AArch64 and execution of CFPRCTX at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the instruction generates a higher priority exception: • Execution of CFP RCTX at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18. • Execution of CFPRCTX at EL0 using AArch32 when EL1 is using AArch64 is trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIVAALE1, bit [47]

Trap execution of TLBI VAALE1 at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VAALE1NXS.

TLBIVAALE1	Meaning
0b0	Execution of TLBI VAALE1 is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAALE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIVALE1, bit [46]

Trap execution of TLBI VALE1 at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VALE1NXS.

TLBIVALE1	Meaning
0b0	Execution of TLBI VALE1 is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VALE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIVAAE1, bit [45]

Trap execution of TLBI VAAE1 at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VAAE1NXS.

TLBIVAAE1	Meaning
0b0	Execution of TLBI VAAE1 is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAAE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIASIDE1, bit [44]

Trap execution of TLBI ASIDE1 at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI ASIDE1NXS.

TLBIASIDE1	Meaning
0b0	Execution of TLBI ASIDE1 is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI ASIDE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIVAE1, bit [43]

Trap execution of TLBI VAE1 at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VAE1NXS.

TLBIVAE1	Meaning
0b0	Execution of TLBI VAE1 is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIVMALLE1, bit [42]

Trap execution of TLBI VMALLE1 at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VMALLE1NXS.

TLBIVMALLE1	Meaning
0b0	Execution of TLBI VMALLE1 is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VMALLE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIRVAALE1, bit [41]

When FEAT_TLBIRANGE is implemented:

Trap execution of TLBI RVAALE1 at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVAALE1NXS.

TLBIRVAALE1	Meaning
0b0	Execution of TLBI RVAALE1 is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAALE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIRVALE1, bit [40]

When FEAT_TLBIRANGE is implemented:

Trap execution of TLBI RVALE1 at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVALE1NXS.

TLBIRVALE1	Meaning
0b0	Execution of TLBI RVALE1 is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVALE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIRVAAE1, bit [39]

When FEAT_TLBIRANGE is implemented:

Trap execution of TLBI RVAAE1 at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVAAE1NXS.

TLBIRVAAE1	Meaning
0b0	Execution of TLBI RVAAE1 is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAAE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIRVAE1, bit [38]

When FEAT_TLBIRANGE is implemented:

Trap execution of TLBI RVAE1 at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVAE1NXS.

TLBIRVAE1	Meaning
0b0	Execution of TLBI RVAE1 is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIRVAALE1IS, bit [37]

When FEAT_TLBIRANGE is implemented:

Trap execution of TLBI RVAALE1IS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVAALE1ISNXS.

TLBIRVAALE1IS	Meaning
0b0	Execution of TLBI RVAALE1IS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIRVALE1IS, bit [36]

When FEAT_TLBIRANGE is implemented:

Trap execution of TLBI RVALE1IS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVALE1ISNXS.

TLBIRVALE1IS	Meaning
0b0	Execution of TLBI RVALE1IS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIRVAAE1IS, bit [35]

When FEAT_TLBIRANGE is implemented:

Trap execution of TLBI RVAAE1IS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVAAE1ISNXS.

TLBIRVAAE1IS	Meaning
0b0	Execution of TLBI RVAAE1IS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAAE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIRVAE1IS, bit [34]

When FEAT_TLBIRANGE is implemented:

Trap execution of TLBI RVAE1IS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVAE1ISNXS.

TLBIRVAE1IS	Meaning
0b0	Execution of TLBI RVAE1IS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIVAALE1IS, bit [33]

Trap execution of TLBI VAALE1IS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VAALE1ISNXS.

TLBIVAALE1IS	Meaning
0b0	Execution of TLBI VAALE1IS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIVALE1IS, bit [32]

Trap execution of TLBI VALE1IS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VALE1ISNXS.

TLBIVALE1IS	Meaning
0b0	Execution of TLBI VALE1IS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIVAAE1IS, bit [31]

Trap execution of TLBI VAAE1IS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VAAE1ISNXS.

TLBIVAAE1IS	Meaning
0b0	Execution of TLBI VAAE1IS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAAE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIASIDE1IS, bit [30]

Trap execution of TLBI ASIDE1IS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI ASIDE1ISNXS.

TLBIASIDE1IS	Meaning
0b0	Execution of TLBI ASIDE1IS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI ASIDE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIVAE1IS, bit [29]

Trap execution of TLBI VAE1IS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VAE1ISNXS.

TLBIVAE1IS	Meaning
0b0	Execution of TLBI VAE1IS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIVMALLE1IS, bit [28]

Trap execution of TLBI VMALLE1IS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VMALLE1ISNXS.

TLBIVMALLE1IS	Meaning
0b0	Execution of TLBI VMALLE1IS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VMALLE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TLBIRVAALE1OS, bit [27]

When FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented:

Trap execution of TLBI RVAALE1OS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVAALE1OSNXS.

TLBIRVAALE1OS	Meaning
0b0	Execution of TLBI RVAALE1OS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAALE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIRVALE1OS, bit [26]

When FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented:

Trap execution of TLBI RVALE1OS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVALE1OSNXS.

TLBIRVALE1OS	Meaning
0b0	Execution of TLBI RVALE1OS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVALE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIRVAAE1OS, bit [25]
When FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented:

Trap execution of TLBI RVAAE1OS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVAAE1OSNXS.

TLBIRVAAE1OS	Meaning
0b0	Execution of TLBI RVAAE1OS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAAE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIRVAE1OS, bit [24]
When FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented:

Trap execution of TLBI RVAE1OS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI RVAE1OSNXS.

TLBIRVAE1OS	Meaning
0b0	Execution of TLBI RVAE1OS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIVAALE1OS, bit [23]
When FEAT_TLBIOS is implemented:

Trap execution of TLBI VAALE1OS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VAALE1OSNXS.

TLBIVAALE1OS	Meaning
0b0	Execution of TLBI VAALE1OS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAALE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIVALE1OS, bit [22]

When FEAT_TLBIOS is implemented:

Trap execution of TLBI VALE1OS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VALE1OSNXS.

TLBIVALE1OS	Meaning
0b0	Execution of TLBI VALE1OS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VALE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIVAAE1OS, bit [21]

When FEAT_TLBIOS is implemented:

Trap execution of TLBI VAAE1OS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VAAE1OSNXS.

TLBIVAAE1OS	Meaning
0b0	Execution of TLBI VAAE1OS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAAE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIASIDE1OS, bit [20]

When FEAT_TLBIOS is implemented:

Trap execution of TLBI ASIDE1OS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI ASIDE1OSNXS.

TLBIASIDE1OS	Meaning
0b0	Execution of TLBI ASIDE1OS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI ASIDE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIVAE1OS, bit [19]

When FEAT_TLBIOS is implemented:

Trap execution of TLBI VAE1OS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VAE1OSNXS.

TLBIVAE1OS	Meaning
0b0	Execution of TLBI VAE1OS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLBIVMALLE1OS, bit [18]

When FEAT_TLBIOS is implemented:

Trap execution of TLBI VMALLE1OS at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and HCRX_EL2.FGTnXS == 0, this field also traps execution of TLBI VMALLE1OSNXS.

TLBIVMALLE1OS	Meaning
0b0	Execution of TLBI VMALLE1OS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VMALLE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATS1E1WP, bit [17]

When FEAT_PAN2 is implemented:

Trap execution of AT S1E1WP at EL1 using AArch64 to EL2.

ATS1E1WP	Meaning
0b0	Execution of AT S1E1WP is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E1WP at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATS1E1RP, bit [16]

When FEAT_PAN2 is implemented:

Trap execution of AT S1E1RP at EL1 using AArch64 to EL2.

ATS1E1RP	Meaning
0b0	Execution of AT S1E1RP is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E1RP at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATS1E0W, bit [15]

Trap execution of AT S1E0W at EL1 using AArch64 to EL2.

ATS1E0W	Meaning
0b0	Execution of AT S1E0W is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E0W at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

ATS1E0R, bit [14]

Trap execution of AT S1E0R at EL1 using AArch64 to EL2.

ATS1E0R	Meaning
0b0	Execution of AT S1E0R is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E0R at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

ATS1E1W, bit [13]

Trap execution of AT S1E1W at EL1 using AArch64 to EL2.

ATS1E1W	Meaning
0b0	Execution of AT S1E1W is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E1W at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

ATS1E1R, bit [12]

Trap execution of AT S1E1R at EL1 using AArch64 to EL2.

ATS1E1R	Meaning
0b0	Execution of AT S1E1R is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E1R at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DCZVA, bit [11]

Trap execution of any of the following AArch64 instructions at EL1 and EL0 to EL2:

- DC ZVA.
- DC GVA, if FEAT_MTE is implemented.
- DC GZVA, if FEAT_MTE is implemented.

Note

Unlike HCR_EL2.TDZ, this field has no effect on DCZID_EL0.DZP.

DCZVA	Meaning
0b0	Execution of the specified instructions is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution at EL1 and EL0 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DCCIVAC, bit [10]

Trap execution of any of the following AArch64 instructions at EL1 and EL0 to EL2:

- DC CIVAC.
- DC CIGVAC, if FEAT_MTE is implemented.
- DC CIGDVAC, if FEAT_MTE is implemented.
- DC CIVAOC, if FEAT_OCCMO is implemented.
- DC CIGDVAOC, if FEAT_OCCMO is implemented.

DCCIVAC	Meaning
0b0	Execution of the specified instructions is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then if the execution can be trapped, execution at EL1 and EL0 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

DCCVADP, bit [9]

When FEAT_DPB2 is implemented:

Trap execution of any of the following AArch64 instructions at EL1 and EL0 to EL2:

- DC CVADP.
- DC CGVADP, if FEAT_MTE is implemented.
- DC CGDVADP, if FEAT_MTE is implemented.

DCCVADP	Meaning
0b0	Execution of the specified instructions is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then if the execution can be trapped, execution at EL1 and EL0 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DCCVAP, bit [8]

Trap execution of any of the following AArch64 instructions at EL1 and EL0 to EL2:

- DC CVAP.
- DC CGVAP, if FEAT_MTE is implemented.
- DC CGDVAP, if FEAT_MTE is implemented.

DCCVAP	Meaning
0b0	Execution of the specified instructions is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then if the execution can be trapped, execution at EL1 and EL0 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DCCVAU, bit [7]

Trap execution of DC CVAU at EL1 and EL0 using AArch64 to EL2.

DCCVAU	Meaning
0b0	Execution of DC CVAU is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then if the execution can be trapped, execution of DC CVAU at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DCCISW, bit [6]

Trap execution of any of the following AArch64 instructions at EL1 to EL2:

- DC CISW.
- DC CIGSW, if FEAT_MTE2 is implemented.
- DC CIGDSW, if FEAT_MTE2 is implemented.

DCCISW	Meaning
0b0	Execution of the specified instructions is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution at EL1 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DCCSW, bit [5]

Trap execution of any of the following AArch64 instructions at EL1 to EL2:

- DC CSW.
- DC CGSW, if FEAT_MTE2 is implemented.
- DC CGDSW, if FEAT_MTE2 is implemented.

DCCSW	Meaning
0b0	Execution of the specified instructions is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution at EL1 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DCISW, bit [4]

Trap execution of any of the following AArch64 instructions at EL1 to EL2:

- DC ISW.
- DC IGSW, if FEAT_MTE2 is implemented.
- DC IGDSW, if FEAT_MTE2 is implemented.

DCISW	Meaning
0b0	Execution of the specified instructions is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution at EL1 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DCIVAC, bit [3]

Trap execution of any of the following AArch64 instructions at EL1 to EL2:

- DC IVAC.
- DC IGVAC, if FEAT_MTE2 is implemented.
- DC IGDVAC, if FEAT_MTE2 is implemented.

DCIVAC	Meaning
0b0	Execution of the specified instructions is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then if the execution can be trapped, execution at EL1 using AArch64 of any of the specified instructions is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

ICIVAU, bit [2]

Trap execution of IC IVAU at EL1 and EL0 using AArch64 to EL2.

ICIVAU	Meaning
0b0	Execution of IC IVAU is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then if the execution can be trapped, execution of IC IVAU at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

ICIALLU, bit [1]

Trap execution of IC IALLU at EL1 using AArch64 to EL2.

ICIALLU	Meaning
0b0	Execution of IC IALLU is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then if the execution can be trapped, execution of IC IALLU at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

ICIALLUIS, bit [0]

Trap execution of IC IALLUIS at EL1 using AArch64 to EL2.

ICIALLUIS	Meaning
0b0	Execution of IC IALLUIS is not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then if the execution can be trapped, execution of IC IALLUIS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HFGITR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HFGITR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b110

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1C8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HFGITR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HFGITR_EL2;
```

```
MSR HFGITR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b110

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1C8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGITR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HFGITR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HFGRTR2_EL2, Hypervisor Fine-Grained Read Trap Register 2

The HFGRTR2_EL2 characteristics are:

Purpose

Provides controls for traps of `MRRS`, `MRS` and `MRC` reads of System registers.

Configuration

This register is present only when FEAT_FGT2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HFGRTR2_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HFGRTR2_EL2 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42
RES0																	nACTLRALIAS_EL1	nACTLRMASK_EL1	nTCR2ALIAS_EL1	nTCRALIAS_EL1	nSCTLRALIAS2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
RES0																					

Bits [63:15]

Reserved, RES0.

nACTLRALIAS_EL1, bit [14]

When FEAT_SRMASK is implemented:

Trap `MRS` reads of ACTLRALIAS_EL1 at EL1 using AArch64 to EL2.

nACTLRALIAS_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then `MRS` reads of ACTLRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.
`0b1`	`MRS` reads of ACTLRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nACTLRMASK_EL1, bit [13]
When FEAT_SRMASK is implemented:

Trap MRS reads of ACTLRMASK_EL1 at EL1 using AArch64 to EL2.

nACTLRMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of ACTLRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of ACTLRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCR2ALIAS_EL1, bit [12]
When FEAT_SRMASK is implemented:

Trap MRS reads of TCR2ALIAS_EL1 at EL1 using AArch64 to EL2.

nTCR2ALIAS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of TCR2ALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of TCR2ALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCRALIAS_EL1, bit [11]

When FEAT_SRMASK is implemented:

Trap MRS reads of TCRALIAS_EL1 at EL1 using AArch64 to EL2.

nTCRALIAS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of TCRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of TCRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLRALIAS2_EL1, bit [10]

When FEAT_SRMASK is implemented:

Trap MRS reads of SCTLRALIAS2_EL1 at EL1 using AArch64 to EL2.

nSCTLRALIAS2_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of SCTLRALIAS2_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SCTLRALIAS2_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLRALIAS_EL1, bit [9]

When FEAT_SRMASK is implemented:

Trap MRS reads of SCTLRALIAS_EL1 at EL1 using AArch64 to EL2.

nSCTLRALIAS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of SCTLRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SCTLRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nCPACRALIAS_EL1, bit [8]

When FEAT_SRMASK is implemented:

Trap MRS reads of CPACRALIAS_EL1 at EL1 using AArch64 to EL2.

nCPACRALIAS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of CPACRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of CPACRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCR2MASK_EL1, bit [7]

When FEAT_SRMASK is implemented:

Trap MRS reads of TCR2MASK_EL1 at EL1 using AArch64 to EL2.

nTCR2MASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of TCR2MASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of TCR2MASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCRMASK_EL1, bit [6]

When FEAT_SRMASK is implemented:

Trap MRS reads of TCRMASK_EL1 at EL1 using AArch64 to EL2.

nTCRMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of TCRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of TCRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLR2MASK_EL1, bit [5]

When FEAT_SRMASK is implemented:

Trap MRS reads of SCTLR2MASK_EL1 at EL1 using AArch64 to EL2.

nSCTLR2MASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of SCTLR2MASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SCTLR2MASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLRMASK_EL1, bit [4]

When FEAT_SRMASK is implemented:

Trap MRS reads of SCTLRMASK_EL1 at EL1 using AArch64 to EL2.

nSCTLRMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of SCTLRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SCTLRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nCPACRMASK_EL1, bit [3]

When FEAT_SRMASK is implemented:

Trap MRS reads of CPACRMASK_EL1 at EL1 using AArch64 to EL2.

nCPACRMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of CPACRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of CPACRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nRCWSMASK_EL1, bit [2]
When FEAT_THE is implemented:

Trap MRS or MRRS reads of RCWSMASK_EL1 at EL1 using AArch64 to EL2.

nRCWSMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then unless the read generates a higher priority exception: • MRS reads of RCWSMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRRS reads of RCWSMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.
0b1	MRS and MRRS reads of RCWSMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nERXGSR_EL1, bit [1]
When FEAT_RASv2 is implemented:

Trap MRS reads of ERXGSR_EL1 at EL1 using AArch64 to EL2.

nERXGSR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of ERXGSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of ERXGSR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPFAR_EL1, bit [0]
When FEAT_PFAR is implemented:

Trap MRS reads of PFAR_EL1 at EL1 using AArch64 to EL2.

nPFAR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MRS reads of PFAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of PFAR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing HFGRTR2_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HFGRTR2_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x2C0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HFGRTR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HFGRTR2_EL2;
```

MSR HFGRTR2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x2C0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGRTR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HFGRTR2_EL2 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HFGRTR_EL2, Hypervisor Fine-Grained Read Trap Register

The HFGRTR_EL2 characteristics are:

Purpose

Provides controls for traps of MRRS, MRS and MRC reads of System registers.

Configuration

This register is present only when FEAT_FGT is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HFGRTR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HFGRTR_EL2 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55
nAMAIR2_EL1	nMAIR2_EL1	nS2POR_EL1	nPOR_EL1	nPOR_EL0	nPIR_EL1	nPIRE0_EL1	nRCWMASK_EL1	nTPIDR2_E
SCXTNUM_EL0	SCXTNUM_EL1	SCTLR_EL1	REVIDR_EL1	PAR_EL1	MPIDR_EL1	MIDR_EL1	MAIR_EL1	LORSA_EL
31	30	29	28	27	26	25	24	23

nAMAIR2_EL1, bit [63]

When FEAT_AIE is implemented:

Trap MRS reads of AMAIR2_EL1 at EL1 using AArch64 to EL2.

nAMAIR2_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of AMAIR2_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of AMAIR2_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nMAIR2_EL1, bit [62]

When FEAT_AIE is implemented:

Trap MRS reads of MAIR2_EL1 at EL1 using AArch64 to EL2.

nMAIR2_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MAIR2_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of MAIR2_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nS2POR_EL1, bit [61]

When FEAT_S2POE is implemented:

Trap MRS reads of S2POR_EL1 at EL1 using AArch64 to EL2.

nS2POR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of S2POR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of S2POR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPOR_EL1, bit [60]

When FEAT_S1POE is implemented:

Trap MRS reads of POR_EL1 at EL1 using AArch64 to EL2.

nPOR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of POR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of POR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPOR_EL0, bit [59]
When FEAT_S1POE is implemented:

Trap MRS reads of POR_EL0 at EL1 and EL0 using AArch64 to EL2.

nPOR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of POR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of POR_EL0 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPIR_EL1, bit [58]
When FEAT_S1PIE is implemented:

Trap MRS reads of PIR_EL1 at EL1 using AArch64 to EL2.

nPIR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of PIR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPIRE0_EL1, bit [57]
When FEAT_S1PIE is implemented:

Trap MRS reads of PIRE0_EL1 at EL1 using AArch64 to EL2.

nPIRE0_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PIRE0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of PIRE0_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nRCWMASK_EL1, bit [56]

When FEAT_THE is implemented:

Trap MRS or MRRS reads of RCWMASK_EL1 at EL1 using AArch64 to EL2.

nRCWMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of RCWMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRRS reads of RCWMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.
0b1	MRS and MRRS reads of RCWMASK_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTPIDR2_EL0, bit [55]

When FEAT_SME is implemented:

Trap MRS reads of TPIDR2_EL0 at EL1 and EL0 using AArch64 to EL2.

nTPIDR2_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TPIDR2_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of TPIDR2_EL0 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSMPRI_EL1, bit [54]
When FEAT_SME is implemented:

Trap MRS reads of SMPRI_EL1 at EL1 using AArch64 to EL2.

nSMPRI_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of SMPRI_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SMPRI_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCS_EL1, bit [53]
When FEAT_GCS is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- GCSCR_EL1.
- GCSPR_EL1.

nGCS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCS_EL0, bit [52]

When FEAT_GCS is implemented:

Trap MRS reads at EL1 and EL0 using AArch64 of any of the following AArch64 System registers to EL2:

- GCSCRE0_EL1, at EL1 only.
- GCSPR_EL0.

nGCS_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 and EL0 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [51]

Reserved, RES0.

nACCDATA_EL1, bit [50]

When FEAT_LS64_ACCDATA is implemented:

Trap MRS reads of ACCDATA_EL1 at EL1 using AArch64 to EL2.

nACCDATA_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ACCDATA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of ACCDATA_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXADDR_EL1, bit [49]

When FEAT_RAS is implemented:

Trap MRS reads of ERXADDR_EL1 at EL1 using AArch64 to EL2.

ERXADDR_EL1	Meaning
0b0	MRS reads of ERXADDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXADDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXPFGCDN_EL1, bit [48]

When FEAT_RASv1p1 is implemented:

Trap MRS reads of ERXPFGCDN_EL1 at EL1 using AArch64 to EL2.

ERXPFGCDN_EL1	Meaning
0b0	MRS reads of ERXPFGCDN_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXPFGCDN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXPFGCTL_EL1, bit [47]

When FEAT_RASv1p1 is implemented:

Trap MRS reads of ERXPFGCTL_EL1 at EL1 using AArch64 to EL2.

ERXPFGCTL_EL1	Meaning
0b0	MRS reads of ERXPFGCTL_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXPFGCTL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXPFGF_EL1, bit [46]
When FEAT_RASv1p1 is implemented:

Trap MRS reads of ERXPFGF_EL1 at EL1 using AArch64 to EL2.

ERXPFGF_EL1	Meaning
0b0	MRS reads of ERXPFGF_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXPFGF_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXMISCn_EL1, bit [45]
When FEAT_RAS is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- ERXMISC0_EL1.
- ERXMISC1_EL1.
- ERXMISC2_EL1.
- ERXMISC3_EL1.

ERXMISCn_EL1	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXSTATUS_EL1, bit [44]

When FEAT_RAS is implemented:

Trap MRS reads of ERXSTATUS_EL1 at EL1 using AArch64 to EL2.

ERXSTATUS_EL1	Meaning
0b0	MRS reads of ERXSTATUS_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXCTLR_EL1, bit [43]

When FEAT_RAS is implemented:

Trap MRS reads of ERXCTLR_EL1 at EL1 using AArch64 to EL2.

ERXCTLR_EL1	Meaning
0b0	MRS reads of ERXCTLR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXFR_EL1, bit [42]

When FEAT_RAS is implemented:

Trap MRS reads of ERXFR_EL1 at EL1 using AArch64 to EL2.

ERXFR_EL1	Meaning
0b0	MRS reads of ERXFR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERRSELR_EL1, bit [41]

When FEAT_RAS is implemented:

Trap MRS reads of ERRSELR_EL1 at EL1 using AArch64 to EL2.

ERRSELR_EL1	Meaning
0b0	MRS reads of ERRSELR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERRSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERRIDR_EL1, bit [40]

When FEAT_RAS is implemented:

Trap MRS reads of ERRIDR_EL1 at EL1 using AArch64 to EL2.

ERRIDR_EL1	Meaning
0b0	MRS reads of ERRIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERRIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ICC_IGRPENn_EL1, bit [39]

When GICv3 is implemented:

Trap MRS reads of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 to EL2.

ICC_IGRPENn_EL1	Meaning
0b0	MRS reads of ICC_IGRPEN<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VBAR_EL1, bit [38]

Trap MRS reads of VBAR_EL1 at EL1 using AArch64 to EL2.

VBAR_EL1	Meaning
0b0	MRS reads of VBAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TTBR1_EL1, bit [37]

Trap MRS or MRRS reads of TTBR1_EL1 at EL1 using AArch64 to EL2.

TTBR1_EL1	Meaning
0b0	MRS and MRRS reads of TTBR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRRS reads of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TTBR0_EL1, bit [36]

Trap MRS or MRRS reads of TTBR0_EL1 at EL1 using AArch64 to EL2.

TTBR0_EL1	Meaning
0b0	MRS and MRRS reads of TTBR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRRS reads of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDR_EL0, bit [35]

Trap MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 to EL2.

TPIDR_EL0	Meaning
0b0	MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURW at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDRRO_EL0, bit [34]

Trap MRS reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURO at EL0 using AArch32 when EL1 is using AArch64 to EL2.

TPIDRRO_EL0	Meaning
0b0	MRS reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURO at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of TPIDRURO at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDR_EL1, bit [33]

Trap MRS reads of TPIDR_EL1 at EL1 using AArch64 to EL2.

TPIDR_EL1	Meaning
0b0	MRS reads of TPIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TCR_EL1, bit [32]

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- TCR_EL1.
- TCR2_EL1, if FEAT_TCR2 is implemented.

TCR_EL1	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SCXTNUM_EL0, bit [31]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MRS reads of SCXTNUM_EL0 at EL1 and EL0 using AArch64 to EL2.

SCXTNUM_EL0	Meaning
0b0	MRS reads of SCXTNUM_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of SCXTNUM_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCXTNUM_EL1, bit [30]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MRS reads of SCXTNUM_EL1 at EL1 using AArch64 to EL2.

SCXTNUM_EL1	Meaning
0b0	MRS reads of SCXTNUM_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of SCXTNUM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCTLR_EL1, bit [29]

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- SCTLR_EL1.
- SCTLR2_EL1, if FEAT_SCTLR2 is implemented.

SCTLR_EL1	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

REVIDR_EL1, bit [28]

Trap MRS reads of REVIDR_EL1 at EL1 using AArch64 to EL2.

REVIDR_EL1	Meaning
0b0	MRS reads of REVIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of REVIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

PAR_EL1, bit [27]

Trap MRS or MRRS reads of PAR_EL1 at EL1 using AArch64 to EL2.

PAR_EL1	Meaning
0b0	MRS and MRRS reads of PAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRRS reads of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

MPIDR_EL1, bit [26]

Trap MRS reads of MPIDR_EL1 at EL1 using AArch64 to EL2.

MPIDR_EL1	Meaning
0b0	MRS reads of MPIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

MIDR_EL1, bit [25]

Trap MRS reads of MIDR_EL1 at EL1 using AArch64 to EL2.

MIDR_EL1	Meaning
0b0	MRS reads of MIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

MAIR_EL1, bit [24]

Trap MRS reads of MAIR_EL1 at EL1 using AArch64 to EL2.

MAIR_EL1	Meaning
0b0	MRS reads of MAIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

LORSA_EL1, bit [23]
When FEAT_LOR is implemented:

Trap MRS reads of LORSA_EL1 at EL1 using AArch64 to EL2.

LORSA_EL1	Meaning
0b0	MRS reads of LORSA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORSA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LORN_EL1, bit [22]
When FEAT_LOR is implemented:

Trap MRS reads of LORN_EL1 at EL1 using AArch64 to EL2.

LORN_EL1	Meaning
0b0	MRS reads of LORN_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LORID_EL1, bit [21]

When FEAT_LOR is implemented:

Trap MRS reads of LORID_EL1 at EL1 using AArch64 to EL2.

LORID_EL1	Meaning
0b0	MRS reads of LORID_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORID_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LOREA_EL1, bit [20]

When FEAT_LOR is implemented:

Trap MRS reads of LOREA_EL1 at EL1 using AArch64 to EL2.

LOREA_EL1	Meaning
0b0	MRS reads of LOREA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LOREA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LORC_EL1, bit [19]

When FEAT_LOR is implemented:

Trap MRS reads of LORC_EL1 at EL1 using AArch64 to EL2.

LORC_EL1	Meaning
0b0	MRS reads of LORC_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORC_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ISR_EL1, bit [18]

Trap MRS reads of ISR_EL1 at EL1 using AArch64 to EL2.

ISR_EL1	Meaning
0b0	MRS reads of ISR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ISR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

FAR_EL1, bit [17]

Trap MRS reads of FAR_EL1 at EL1 using AArch64 to EL2.

FAR_EL1	Meaning
0b0	MRS reads of FAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of FAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

ESR_EL1, bit [16]

Trap MRS reads of ESR_EL1 at EL1 using AArch64 to EL2.

ESR_EL1	Meaning
0b0	MRS reads of ESR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ESR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DCZID_EL0, bit [15]

Trap MRS reads of DCZID_EL0 at EL1 and EL0 using AArch64 to EL2.

DCZID_EL0	Meaning
0b0	MRS reads of DCZID_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DCZID_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CTR_EL0, bit [14]

Trap MRS reads of CTR_EL0 at EL1 and EL0 using AArch64 to EL2.

CTR_EL0	Meaning
0b0	MRS reads of CTR_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CSSELR_EL1, bit [13]

Trap MRS reads of CSSELR_EL1 at EL1 using AArch64 to EL2.

CSSELR_EL1	Meaning
0b0	MRS reads of CSSELR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CSSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CPACR_EL1, bit [12]

Trap MRS reads of CPACR_EL1 at EL1 using AArch64 to EL2.

CPACR_EL1	Meaning
0b0	MRS reads of CPACR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CPACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CONTEXTIDR_EL1, bit [11]

Trap MRS reads of CONTEXTIDR_EL1 at EL1 using AArch64 to EL2.

CONTEXTIDR_EL1	Meaning
0b0	MRS reads of CONTEXTIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CONTEXTIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CLIDR_EL1, bit [10]

Trap MRS reads of CLIDR_EL1 at EL1 using AArch64 to EL2.

CLIDR_EL1	Meaning
0b0	MRS reads of CLIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CLIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CCSIDR_EL1, bit [9]

Trap MRS reads of CCSIDR_EL1 at EL1 using AArch64 to EL2.

CCSIDR_EL1	Meaning
0b0	MRS reads of CCSIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CCSIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

APIBKey, bit [8]

When FEAT_PAuth is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIBKeyHi_EL1.
- APIBKeyLo_EL1.

APIBKey	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APIAKey, bit [7]

When FEAT_PAuth is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIAKeyHi_EL1.
- APIAKeyLo_EL1.

APIAKey	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APGAKey, bit [6]

When FEAT_PAuth is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APGAKeyHi_EL1.
- APGAKeyLo_EL1.

APGAKey	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APDBKey, bit [5]
When FEAT_PAuth is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDBKeyHi_EL1.
- APDBKeyLo_EL1.

APDBKey	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APDAKey, bit [4]
When FEAT_PAuth is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDAKeyHi_EL1.
- APDAKeyLo_EL1.

APDAKey	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMAIR_EL1, bit [3]

Trap MRS reads of AMAIR_EL1 at EL1 using AArch64 to EL2.

AMAIR_EL1	Meaning
0b0	MRS reads of AMAIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of AMAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AIDR_EL1, bit [2]

Trap MRS reads of AIDR_EL1 at EL1 using AArch64 to EL2.

AIDR_EL1	Meaning
0b0	MRS reads of AIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of AIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AFSR1_EL1, bit [1]

Trap MRS reads of AFSR1_EL1 at EL1 using AArch64 to EL2.

AFSR1_EL1	Meaning
0b0	MRS reads of AFSR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of AFSR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AFSR0_EL1, bit [0]

Trap MRS reads of AFSR0_EL1 at EL1 using AArch64 to EL2.

AFSR0_EL1	Meaning
0b0	MRS reads of AFSR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of AFSR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HFGRTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HFGRTR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1B8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HFGRTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HFGRTR_EL2;
```

```
MSR HFGRTR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1B8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGRTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HFGRTR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HFGWTR2_EL2, Hypervisor Fine-Grained Write Trap Register 2

The HFGWTR2_EL2 characteristics are:

Purpose

Provides controls for traps of `MSRR`, `MSR` and `MCR` writes of System registers.

Configuration

This register is present only when FEAT_FGT2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HFGWTR2_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HFGWTR2_EL2 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42
RES0																	nACTLRALIAS_EL1	nACTLRMASK_EL1	nTCR2ALIAS_EL1	nTCRALIAS_EL1	nSCTLRALIAS2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
RES0																					

Bits [63:15]

Reserved, RES0.

nACTLRALIAS_EL1, bit [14]

When FEAT_SRMASK is implemented:

Trap `MSR` writes of ACTLRALIAS_EL1 at EL1 using AArch64 to EL2.

nACTLRALIAS_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then `MSR` writes of ACTLRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	`MSR` writes of ACTLRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nACTLRMASK_EL1, bit [13]
When FEAT_SRMASK is implemented:

Trap MSR writes of ACTLRMASK_EL1 at EL1 using AArch64 to EL2.

nACTLRMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of ACTLRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of ACTLRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCR2ALIAS_EL1, bit [12]
When FEAT_SRMASK is implemented:

Trap MSR writes of TCR2ALIAS_EL1 at EL1 using AArch64 to EL2.

nTCR2ALIAS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of TCR2ALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of TCR2ALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCRALIAS_EL1, bit [11]

When FEAT_SRMASK is implemented:

Trap MSR writes of TCRALIAS_EL1 at EL1 using AArch64 to EL2.

nTCRALIAS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of TCRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of TCRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLRALIAS2_EL1, bit [10]

When FEAT_SRMASK is implemented:

Trap MSR writes of SCTLRALIAS2_EL1 at EL1 using AArch64 to EL2.

nSCTLRALIAS2_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of SCTLRALIAS2_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SCTLRALIAS2_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLRALIAS_EL1, bit [9]

When FEAT_SRMASK is implemented:

Trap MSR writes of SCTLRALIAS_EL1 at EL1 using AArch64 to EL2.

nSCTLRALIAS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of SCTLRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SCTLRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nCPACRALIAS_EL1, bit [8]

When FEAT_SRMASK is implemented:

Trap MSR writes of CPACRALIAS_EL1 at EL1 using AArch64 to EL2.

nCPACRALIAS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of CPACRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of CPACRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCR2MASK_EL1, bit [7]

When FEAT_SRMASK is implemented:

Trap MSR writes of TCR2MASK_EL1 at EL1 using AArch64 to EL2.

nTCR2MASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of TCR2MASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of TCR2MASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCRMASK_EL1, bit [6]

When FEAT_SRMASK is implemented:

Trap MSR writes of TCRMASK_EL1 at EL1 using AArch64 to EL2.

nTCRMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of TCRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of TCRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLR2MASK_EL1, bit [5]

When FEAT_SRMASK is implemented:

Trap MSR writes of SCTLR2MASK_EL1 at EL1 using AArch64 to EL2.

nSCTLR2MASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of SCTLR2MASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SCTLR2MASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLRMASK_EL1, bit [4]

When FEAT_SRMASK is implemented:

Trap MSR writes of SCTLRMASK_EL1 at EL1 using AArch64 to EL2.

nSCTLRMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of SCTLRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SCTLRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nCPACRMASK_EL1, bit [3]

When FEAT_SRMASK is implemented:

Trap MSR writes of CPACRMASK_EL1 at EL1 using AArch64 to EL2.

nCPACRMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of CPACRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of CPACRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nRCWSMASK_EL1, bit [2]

When FEAT_THE is implemented:

Trap MSR or MSRR writes of RCWSMASK_EL1 at EL1 using AArch64 to EL2.

nRCWSMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then unless the write generates a higher priority exception: • MSR writes of RCWSMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MSRR writes of RCWSMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.
0b1	MSR and MSRR writes of RCWSMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [1]

Reserved, RES0.

nPFAR_EL1, bit [0]

When FEAT_PFAR is implemented:

Trap MSR writes of PFAR_EL1 at EL1 using AArch64 to EL2.

nPFAR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of PFAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PFAR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing HFGWTR2_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HFGWTR2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b011

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x2C8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HFGWTR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HFGWTR2_EL2;
```

```
MSR HFGWTR2_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b011

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x2C8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGWTR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HFGWTR2_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HFGWTR_EL2, Hypervisor Fine-Grained Write Trap Register

The HFGWTR_EL2 characteristics are:

Purpose

Provides controls for traps of MSRR, MSR and MCR writes of System registers.

Configuration

This register is present only when FEAT_FGT is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HFGWTR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HFGWTR_EL2 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	
nAMAIR2_EL1	nMAIR2_EL1	nS2POR_EL1	nPOR_EL1	nPOR_EL0	nPIR_EL1	nPIRE0_EL1	nRCWMASK_EL1	nTPIDR2_EL0	nS
SCXTNUM_EL0	SCXTNUM_EL1	SCTLR_EL1	RES0	PAR_EL1	RES0		MAIR_EL1	LORSA_EL1	L
31	30	29	28	27	26	25	24	23	

nAMAIR2_EL1, bit [63]

When FEAT_AIE is implemented:

Trap MSR writes of AMAIR2_EL1 at EL1 using AArch64 to EL2.

nAMAIR2_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AMAIR2_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of AMAIR2_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nMAIR2_EL1, bit [62]

When FEAT_AIE is implemented:

Trap MSR writes of MAIR2_EL1 at EL1 using AArch64 to EL2.

nMAIR2_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of MAIR2_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of MAIR2_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nS2POR_EL1, bit [61]

When FEAT_S2POE is implemented:

Trap MSR writes of S2POR_EL1 at EL1 using AArch64 to EL2.

nS2POR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of S2POR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of S2POR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPOR_EL1, bit [60]

When FEAT_S1POE is implemented:

Trap MSR writes of POR_EL1 at EL1 using AArch64 to EL2.

nPOR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of POR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of POR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPOR_EL0, bit [59]
When FEAT_S1POE is implemented:

Trap MSR writes of POR_EL0 at EL1 and EL0 using AArch64 to EL2.

nPOR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of POR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of POR_EL0 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPIR_EL1, bit [58]
When FEAT_S1PIE is implemented:

Trap MSR writes of PIR_EL1 at EL1 using AArch64 to EL2.

nPIR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PIR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPIRE0_EL1, bit [57]
When FEAT_S1PIE is implemented:

Trap MSR writes of PIRE0_EL1 at EL1 using AArch64 to EL2.

nPIRE0_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PIRE0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PIRE0_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nRCWMASK_EL1, bit [56]

When FEAT_THE is implemented:

Trap MSR or MSRR writes of RCWMASK_EL1 at EL1 using AArch64 to EL2.

nRCWMASK_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of RCWMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MSRR writes of RCWMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.
0b1	MSR and MSRR writes of RCWMASK_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTPIDR2_EL0, bit [55]

When FEAT_SME is implemented:

Trap MSR writes of TPIDR2_EL0 at EL1 and EL0 using AArch64 to EL2.

nTPIDR2_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TPIDR2_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of TPIDR2_EL0 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSMPRI_EL1, bit [54]
When FEAT_SME is implemented:

Trap MSR writes of SMPRI_EL1 at EL1 using AArch64 to EL2.

nSMPRI_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SMPRI_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SMPRI_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCS_EL1, bit [53]
When FEAT_GCS is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- GCSCR_EL1.
- GCSPR_EL1.

nGCS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCS_EL0, bit [52]

When FEAT_GCS is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- GCSCRE0_EL1.
- GCSPR_EL0.

nGCS_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [51]

Reserved, RES0.

nACCDATA_EL1, bit [50]

When FEAT_LS64_ACCDATA is implemented:

Trap MSR writes of ACCDATA_EL1 at EL1 using AArch64 to EL2.

nACCDATA_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ACCDATA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of ACCDATA_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXADDR_EL1, bit [49]

When FEAT_RAS is implemented:

Trap MSR writes of ERXADDR_EL1 at EL1 using AArch64 to EL2.

ERXADDR_EL1	Meaning
0b0	MSR writes of ERXADDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXADDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXPFGCDN_EL1, bit [48]

When FEAT_RASv1p1 is implemented:

Trap MSR writes of ERXPFGCDN_EL1 at EL1 using AArch64 to EL2.

ERXPFGCDN_EL1	Meaning
0b0	MSR writes of ERXPFGCDN_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXPFGCDN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXPFGCTL_EL1, bit [47]

When FEAT_RASv1p1 is implemented:

Trap MSR writes of ERXPFGCTL_EL1 at EL1 using AArch64 to EL2.

ERXPFGCTL_EL1	Meaning
0b0	MSR writes of ERXPFGCTL_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXPFGCTL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [46]

Reserved, RES0.

ERXMISCn_EL1, bit [45]

When FEAT_RAS is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- ERXMISC0_EL1.
- ERXMISC1_EL1.
- ERXMISC2_EL1.
- ERXMISC3_EL1.

ERXMISCn_EL1	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXSTATUS_EL1, bit [44]

When FEAT_RAS is implemented:

Trap MSR writes of ERXSTATUS_EL1 at EL1 using AArch64 to EL2.

ERXSTATUS_EL1	Meaning
0b0	MSR writes of ERXSTATUS_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXCTLR_EL1, bit [43]

When FEAT_RAS is implemented:

Trap MSR writes of ERXCTLR_EL1 at EL1 using AArch64 to EL2.

ERXCTLR_EL1	Meaning
0b0	MSR writes of ERXCTLR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

ERRSELR_EL1, bit [41]

When FEAT_RAS is implemented:

Trap MSR writes of ERRSELR_EL1 at EL1 using AArch64 to EL2.

ERRSELR_EL1	Meaning
0b0	MSR writes of ERRSELR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERRSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [40]

Reserved, RES0.

ICC_IGRPENn_EL1, bit [39]

When GICv3 is implemented:

Trap MSR writes of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 to EL2.

ICC_IGRPENn_EL1	Meaning
0b0	MSR writes of ICC_IGRPEN<n>_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VBAR_EL1, bit [38]

Trap MSR writes of VBAR_EL1 at EL1 using AArch64 to EL2.

VBAR_EL1	Meaning
0b0	MSR writes of VBAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TTBR1_EL1, bit [37]

Trap MSR or MSRR writes of TTBR1_EL1 at EL1 using AArch64 to EL2.

TTBR1_EL1	Meaning
0b0	MSR and MSRR writes of TTBR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MSRR writes of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TTBR0_EL1, bit [36]

Trap MSR or MSRR writes of TTBR0_EL1 at EL1 using AArch64 to EL2.

TTBR0_EL1	Meaning
0b0	MSR and MSRR writes of TTBR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MSRR writes of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDR_EL0, bit [35]

Trap MSR writes of TPIDR_EL0 at EL1 and EL0 using AArch64 and MCR writes of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 to EL2.

TPIDR_EL0	Meaning
0b0	MSR writes of TPIDR_EL0 at EL1 and EL0 using AArch64 and MCR writes of TPIDRURW at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of TPIDR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDRRO_EL0, bit [34]

Trap MSR writes of TPIDRRO_EL0 at EL1 using AArch64 to EL2.

TPIDRRO_EL0	Meaning
0b0	MSR writes of TPIDRRO_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TPIDRRO_EL0 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDR_EL1, bit [33]

Trap MSR writes of TPIDR_EL1 at EL1 using AArch64 to EL2.

TPIDR_EL1	Meaning
0b0	MSR writes of TPIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TCR_EL1, bit [32]

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- TCR_EL1.
- TCR2_EL1, if FEAT_TCR2 is implemented.

TCR_EL1	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SCXTNUM_EL0, bit [31]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MSR writes of SCXTNUM_EL0 at EL1 and EL0 using AArch64 to EL2.

SCXTNUM_EL0	Meaning
0b0	MSR writes of SCXTNUM_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SCXTNUM_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCXTNUM_EL1, bit [30]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MSR writes of SCXTNUM_EL1 at EL1 using AArch64 to EL2.

SCXTNUM_EL1	Meaning
0b0	MSR writes of SCXTNUM_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SCXTNUM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCTLR_EL1, bit [29]

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- SCTLR_EL1.
- SCTLR2_EL1, if FEAT_SCTLR2 is implemented.

SCTLR_EL1	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [28]

Reserved, RES0.

PAR_EL1, bit [27]

Trap MSR or MSRR writes of PAR_EL1 at EL1 using AArch64 to EL2.

PAR_EL1	Meaning
0b0	MSR and MSRR writes of PAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MSRR writes of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [26:25]

Reserved, RES0.

MAIR_EL1, bit [24]

Trap MSR writes of MAIR_EL1 at EL1 using AArch64 to EL2.

MAIR_EL1	Meaning
0b0	MSR writes of MAIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of MAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

LORSA_EL1, bit [23]

When FEAT_LOR is implemented:

Trap MSR writes of LORSA_EL1 at EL1 using AArch64 to EL2.

LORSA_EL1	Meaning
0b0	MSR writes of LORSA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LORSA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LORN_EL1, bit [22]

When FEAT_LOR is implemented:

Trap MSR writes of LORN_EL1 at EL1 using AArch64 to EL2.

LORN_EL1	Meaning
0b0	MSR writes of LORN_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LORN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [21]

Reserved, RES0.

LOREA_EL1, bit [20]

When FEAT_LOR is implemented:

Trap MSR writes of LOREA_EL1 at EL1 using AArch64 to EL2.

LOREA_EL1	Meaning
0b0	MSR writes of LOREA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LOREA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LORC_EL1, bit [19]

When FEAT_LOR is implemented:

Trap MSR writes of LORC_EL1 at EL1 using AArch64 to EL2.

LORC_EL1	Meaning
0b0	MSR writes of LORC_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LORC_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

FAR_EL1, bit [17]

Trap MSR writes of FAR_EL1 at EL1 using AArch64 to EL2.

FAR_EL1	Meaning
0b0	MSR writes of FAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of FAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

- Otherwise, this field resets to an architecturally UNKNOWN value.

ESR_EL1, bit [16]

Trap MSR writes of ESR_EL1 at EL1 using AArch64 to EL2.

ESR_EL1	Meaning
0b0	MSR writes of ESR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ESR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [15:14]

Reserved, RES0.

CSSELR_EL1, bit [13]

Trap MSR writes of CSSELR_EL1 at EL1 using AArch64 to EL2.

CSSELR_EL1	Meaning
0b0	MSR writes of CSSELR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of CSSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CPACR_EL1, bit [12]

Trap MSR writes of CPACR_EL1 at EL1 using AArch64 to EL2.

CPACR_EL1	Meaning
0b0	MSR writes of CPACR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of CPACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CONTEXTIDR_EL1, bit [11]

Trap MSR writes of CONTEXTIDR_EL1 at EL1 using AArch64 to EL2.

CONTEXTIDR_EL1	Meaning
0b0	MSR writes of CONTEXTIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of CONTEXTIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [10:9]

Reserved, RES0.

APIBKey, bit [8]

When FEAT_PAuth is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIBKeyHi_EL1.
- APIBKeyLo_EL1.

APIBKey	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APIAKey, bit [7]

When FEAT_PAuth is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIAKeyHi_EL1.
- APIAKeyLo_EL1.

APIAKey	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APGAKey, bit [6]

When FEAT_PAuth is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APGAKeyHi_EL1.
- APGAKeyLo_EL1.

APGAKey	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APDBKey, bit [5]

When FEAT_PAuth is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDBKeyHi_EL1.
- APDBKeyLo_EL1.

APDBKey	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APDAKey, bit [4]

When FEAT_PAuth is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDAKeyHi_EL1.
- APDAKeyLo_EL1.

APDAKey	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMAIR_EL1, bit [3]

Trap MSR writes of AMAIR_EL1 at EL1 using AArch64 to EL2.

AMAIR_EL1	Meaning
0b0	MSR writes of AMAIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AMAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

AFSR1_EL1, bit [1]

Trap MSR writes of AFSR1_EL1 at EL1 using AArch64 to EL2.

AFSR1_EL1	Meaning
0b0	MSR writes of AFSR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AFSR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AFSR0_EL1, bit [0]

Trap MSR writes of AFSR0_EL1 at EL1 using AArch64 to EL2.

AFSR0_EL1	Meaning
0b0	MSR writes of AFSR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AFSR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HFGWTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HFGWTR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1C0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HFGWTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HFGWTR_EL2;
```

```
MSR HFGWTR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1C0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGWTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HFGWTR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HPFAR_EL2, Hypervisor IPA Fault Address Register

The HPFAR_EL2 characteristics are:

Purpose

Holds the faulting IPA for some aborts on a stage 2 translation taken to EL2 and GPC exceptions due to a fault on an access for a stage 2 translation.

Configuration

AArch64 System register HPFAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HPFAR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to HPFAR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

The HPFAR_EL2 is written for:

- A Translation fault, Access Flag fault, or Address Size fault on a stage 2 translation not on a stage 1 translation table walk.
- A Translation fault, Access Flag fault, Address Size fault, or Permission fault on stage 2 translation of an address accessed in a stage 1 translation table walk.
- If FEAT_RME is implemented, a Granule Protection Check fault in the second stage of translation.

For all other exceptions taken to EL2, this register is UNKNOWN.

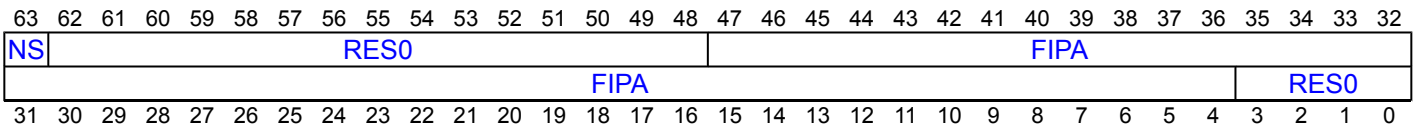
Note

The address held in this register is an address accessed by the instruction fetch or data access that caused the exception that gave rise to the Instruction Abort exception or Data Abort exception. It is the lower address that gave rise to the fault that is reported. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize which fault is reported.

Attributes

HPFAR_EL2 is a 64-bit register.

Field descriptions



Execution at EL1 or EL0 makes HPFAR_EL2 become UNKNOWN.

NS, bit [63]

When FEAT_SEL2 is implemented:

Faulting IPA address space.

NS	Meaning
0b0	Faulting IPA is from the Secure IPA space.
0b1	Faulting IPA is from the Non-secure IPA space.

For Data Abort exceptions or Instruction Abort exceptions taken to Non-secure EL2:

- This field is RES0.
- The address is from the Non-secure IPA space.

If FEAT_RME is implemented, for Data Abort exceptions or Instruction Abort exceptions taken to Realm EL2:

- This field is RES0.
- The address is from the Realm IPA space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

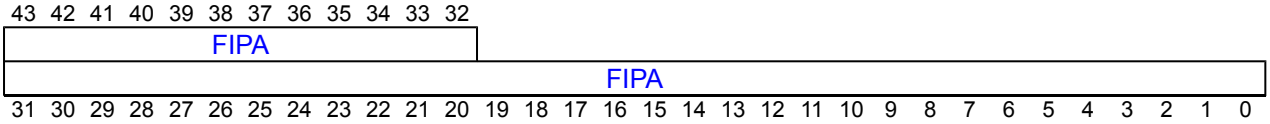
Reserved, RES0.

Bits [62:48]

Reserved, RES0.

FIPA, bits [47:4]

FIPA encoding when FEAT_D128 is implemented



FIPA, bits [43:0]

Bits [55:12] of the Faulting Intermediate Physical Address.

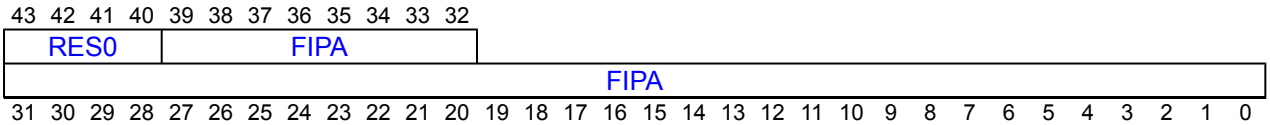
For implementations with fewer than 55 physical address bits, the corresponding upper bits in this field are RES0.

When FEAT_MOPS is implemented, the value presented in FIPA on a synchronous exception that set the HPFAR_EL2 from any of the Memory Copy and Memory Set instructions is within the address range of the current stage 2 translation granule, aligned to the size of the current stage 2 translation granule, of the address that generated the Data abort. In this case, bits[(n-1):0] of the value are UNKNOWN, where 2^n is the current stage 2 translation granule size in bytes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FIPA encoding when FEAT_LPA is implemented and FEAT_D128 is not implemented



Bits [43:40]

Reserved, RES0.

FIPA, bits [39:0]

Bits [51:12] of the Faulting Intermediate Physical Address.

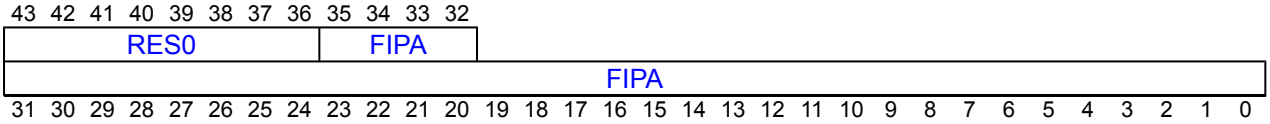
For implementations with fewer than 52 physical address bits, the corresponding upper bits in this field are RES0.

When FEAT_MOPS is implemented, the value presented in FIPA on a synchronous exception that set the HPFAR_EL2 from any of the Memory Copy and Memory Set instructions is within the address range of the current stage 2 translation granule, aligned to the size of the current stage 2 translation granule, of the address that generated the Data abort. In this case, bits[(n-1):0] of the value are UNKNOWN, where 2^n is the current stage 2 translation granule size in bytes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FIPA encoding when FEAT_LPA is not implemented



Bits [43:36]

Reserved, RES0.

FIPA, bits [35:0]

Bits[47:12] Faulting Intermediate Physical Address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

When FEAT_MOPS is implemented, the value presented in FIPA on a synchronous exception that set the HPFAR_EL2 from any of the Memory Copy and Memory Set instructions is within the address range of the current stage 2 translation granule, aligned to the size of the current stage 2 translation granule, of the address that generated the Data abort. In this case, bits[(n-1):0] of the value are UNKNOWN, where 2^n is the current stage 2 translation granule size in bytes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:0]

Reserved, RES0.

Accessing HPFAR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HPFAR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = HPFAR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HPFAR_EL2;
```

```
MSR HPFAR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HPFAR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HPFAR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HSTR_EL2, Hypervisor System Trap Register

The HSTR_EL2 characteristics are:

Purpose

Controls trapping to EL2 of EL1 or lower AArch32 accesses to the System register in the coproc == `0b1111` encoding space, by the CRn value used to access the register using MCR or MRC instruction. When the register is accessible using an MCRR or MRRC instruction, this is the CRm value used to access the register.

Configuration

AArch64 System register HSTR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HSTR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to HSTR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

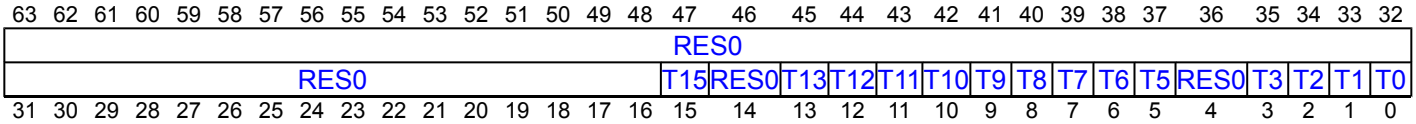
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

HSTR_EL2 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:16, 14, 4]

Reserved, RES0.

T<n>, bit [n], for n = 15, 13 to 5, 3 to 0

The remaining fields control whether EL0 and EL1 accesses, using MCR, MRC, MCRR, and MRRC instructions, to the System registers in the coproc == `0b1111` encoding space, are trapped to EL2 as follows:

- MCR or MRC accesses to these registers that are trapped to EL2 are reported using EC syndrome value `0x03`, unless the access is UNDEFINED.
- MCRR or MRRC accesses to these registers that are trapped to EL2 are reported using EC syndrome value `0x04`, unless the access is UNDEFINED.

T<n>	Meaning
0b0	This control has no effect on EL0 or EL1 accesses to System registers.
0b1	System registers in the coproc == 0b1111 encoding space and CRn == <n> or CRm == <n> where T<n> is the name of this field, are trapped as follows: • An EL1 MCR or MRC access is trapped to EL2. • An EL0 MCR or MRC access is trapped to EL2, if the access is not UNDEFINED when the value of this field is 0. • An EL1 MCRR or MRRC access is trapped to EL2. • An EL0 MCRR or MRRC access is trapped to EL2, if the access is not UNDEFINED when the value of this field is 0. It is IMPLEMENTATION DEFINED whether an EL0 access using AArch32 is trapped to EL2, or is UNDEFINED. If the access is UNDEFINED, and generates an exception that is taken to EL1 or EL2 using AArch64, this is reported with EC syndrome value 0x00. **Note** Arm expects that trapping to EL2 of EL0 accesses to these registers is unusual and used only when the hypervisor must virtualize EL0 operation. Arm recommends that, whenever possible, EL0 accesses to these registers behave as they would if the implementation did not include EL2. This means that, if the architecture does not support the EL0 access, then the register access instruction is treated as UNDEFINED and generates an exception that is taken to EL1.

For example, when HSTR_EL2.T7 is 1, for instructions executed at EL1:

- An MCR or MRC instruction with coproc set to 0b1111 and <CRn> set to c7 is trapped to EL2.
- An MCRR or MRRC instruction with coproc set to 0b1111 and <CRm> set to c7 is trapped to EL2.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
RES0																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Reserved, RES0.

Accessing HSTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HSTR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b011

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x080];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = HSTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HSTR_EL2;
```

```
MSR HSTR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b011

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x080] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HSTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HSTR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

IC IALLU, Instruction Cache Invalidate All to PoU

The IC IALLU characteristics are:

Purpose

Invalidate all instruction caches of the PE executing the instruction to the Point of Unification.

Configuration

AArch64 System instruction IC IALLU performs the same function as AArch32 System instruction ICIALLU.

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to IC IALLU are UNDEFINED.

Attributes

IC IALLU is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing IC IALLU

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
IC IALLU{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TOCU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.ICIALLU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.IC(CacheOpScope_ALLUIS);
    else
        AArch64.IC(CacheOpScope_ALLU);
elsif PSTATE.EL == EL2 then
    AArch64.IC(CacheOpScope_ALLU);
elsif PSTATE.EL == EL3 then
    AArch64.IC(CacheOpScope_ALLU);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

IC IALLUIS, Instruction Cache Invalidate All to PoU, Inner Shareable

The IC IALLUIS characteristics are:

Purpose

Invalidate all instruction caches in the Inner Shareable domain of the PE executing the instruction to the Point of Unification.

Configuration

AArch64 System instruction IC IALLUIS performs the same function as AArch32 System instruction ICIALLUIS.

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to IC IALLUIS are UNDEFINED.

Attributes

IC IALLUIS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing IC IALLUIS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
IC IALLUIS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b0111	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TICAB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.ICIALLUIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.IC(CacheOpScope_ALLUIS);
elsif PSTATE.EL == EL2 then
    AArch64.IC(CacheOpScope_ALLUIS);
elsif PSTATE.EL == EL3 then
    AArch64.IC(CacheOpScope_ALLUIS);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

IC IVAU, Instruction Cache line Invalidate by VA to PoU

The IC IVAU characteristics are:

Purpose

Invalidate instruction cache by address to Point of Unification.

Configuration

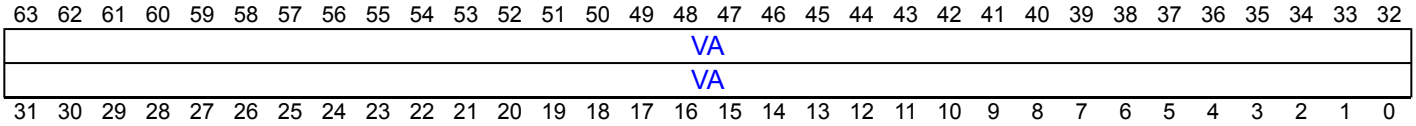
AArch64 System instruction IC IVAU performs the same function as AArch32 System instruction ICIMVAU.

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to IC IVAU are UNDEFINED.

Attributes

IC IVAU is a 64-bit System instruction.

Field descriptions



VA, bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing IC IVAU

If EL0 access is enabled, when executed at EL0, the instruction may generate a Permission fault, subject to the constraints described in 'MMU faults generated by cache maintenance operations'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
IC IVAU{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TOCU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.ICIVAU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.IC(X[t, 64], CacheOpScope_PoU);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TOCU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.ICIVAU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.IC(X[t, 64], CacheOpScope_PoU);
elsif PSTATE.EL == EL2 then
    AArch64.IC(X[t, 64], CacheOpScope_PoU);
elsif PSTATE.EL == EL3 then
    AArch64.IC(X[t, 64], CacheOpScope_PoU);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_AP0R<n>_EL1, Interrupt Controller Active Priorities Group 0 Registers, n = 0 - 3

The ICC_AP0R<n>_EL1 characteristics are:

Purpose

Provides information about Group 0 active priorities.

Configuration

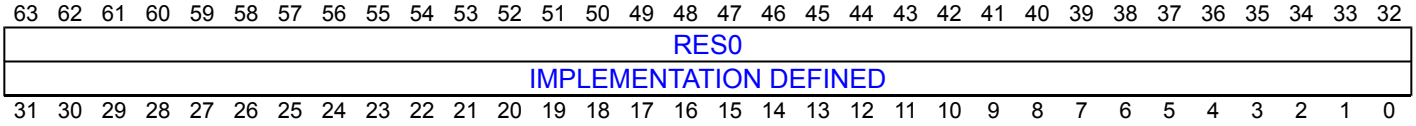
AArch64 System register ICC_AP0R<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICC_AP0R<n>[31:0].

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_AP0R<n>_EL1 are UNDEFINED.

Attributes

ICC_AP0R<n>_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'00000000000000000000000000000000'`.

Additional information

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value `0x00000000` is consistent with no interrupts being active.

Accessing ICC_AP0R<n>_EL1

Writing to these registers with any value other than the last read value of the register (or `0x00000000` when there are no Group 0 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP0R1_EL1 is implemented only in implementations that support 6 or more bits of priority. ICC_AP0R2_EL1 and ICC_AP0R3_EL1 are implemented only in implementations that support 7 or more bits of priority. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR_EL2.PREbits.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICC_AP0R<n>_EL1.
- Secure ICC_AP1R<n>_EL1.
- Non-secure ICC_AP1R<n>_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_AP0R<m>_EL1 ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b1:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_AP0R_EL1[m];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_AP0R_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_AP0R_EL1[m];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_AP0R_EL1[m];
```

```
MSR ICC_AP0R<m>_EL1, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2

0b11	0b000	0b1100	0b1000	0b1:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_AP0R_EL1[m] = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_AP1R<n>_EL1, Interrupt Controller Active Priorities Group 1 Registers, n = 0 - 3

The ICC_AP1R<n>_EL1 characteristics are:

Purpose

Provides information about Group 1 active priorities.

Configuration

This register is banked between ICC_AP1R<n>_EL1 and ICC_AP1R<n>_EL1_S and ICC_AP1R<n>_EL1_NS.

AArch64 System register ICC_AP1R<n>_EL1 bits [31:0] (ICC_AP1R<n>_EL1_S) are architecturally mapped to AArch32 System register ICC_AP1R<n>[31:0] (ICC_AP1R<n>_S).

AArch64 System register ICC_AP1R<n>_EL1 bits [31:0] (ICC_AP1R<n>_EL1_NS) are architecturally mapped to AArch32 System register ICC_AP1R<n>[31:0] (ICC_AP1R<n>_NS).

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_AP1R<n>_EL1 are UNDEFINED.

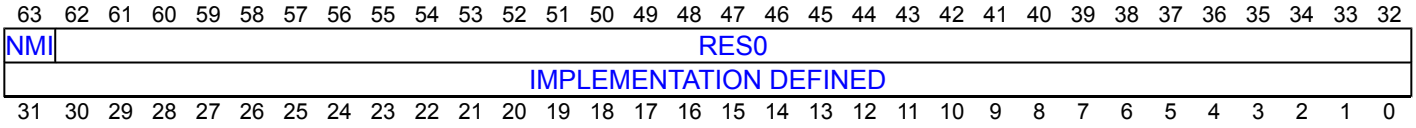
Attributes

ICC_AP1R<n>_EL1 is a 64-bit register.

This register has the following instances:

- ICC_AP1R<n>_EL1, when EL3 is not implemented.
- ICC_AP1R<n>_EL1_S, when EL3 is implemented.
- ICC_AP1R<n>_EL1_NS, when EL3 is implemented.

Field descriptions



NMI, bit [63]

When FEAT_GICv3_NMI is implemented and n == 0:

Indicates whether there is an active NMI priority.

NMI	Meaning
0b0	There is no active Group 1 NMI, or all active Group 1 NMIs have undergone priority drop.
0b1	There is an active Group 1 NMI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bits [62:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00000000000000000000000000000000'.

Additional information

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICC_AP1R<n>_EL1

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP1R1_EL1 is implemented only in implementations that support 6 or more bits of priority. ICC_AP1R2_EL1 and ICC_AP1R3_EL1 are implemented only in implementations that support 7 or more bits of priority. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR_EL2.PREbits.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICC_AP0R<n>_EL1.
- Secure ICC_AP1R<n>_EL1.
- Non-secure ICC_AP1R<n>_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_AP1R<m>_EL1 ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_AP1R_EL1[m];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_AP1R_EL1_S[m];
        else
            X[t, 64] = ICC_AP1R_EL1_NS[m];
    else
        X[t, 64] = ICC_AP1R_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_AP1R_EL1_S[m];
        else
            X[t, 64] = ICC_AP1R_EL1_NS[m];
    else
        X[t, 64] = ICC_AP1R_EL1[m];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_AP1R_EL1_S[m];
        else
            X[t, 64] = ICC_AP1R_EL1_NS[m];
```

```
MSR ICC_AP1R<m>_EL1, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_AP1R_EL1[m] = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[m] = X[t, 64];
        else
            ICC_AP1R_EL1_NS[m] = X[t, 64];
    else
        ICC_AP1R_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[m] = X[t, 64];
        else
            ICC_AP1R_EL1_NS[m] = X[t, 64];
    else
        ICC_AP1R_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[m] = X[t, 64];
        else
            ICC_AP1R_EL1_NS[m] = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_ASGI1R_EL1, Interrupt Controller Alias Software Generated Interrupt Group 1 Register

The ICC_ASGI1R_EL1 characteristics are:

Purpose

Generates Group 1 SGIs for the Security state that is not the current Security state.

Configuration

AArch64 System register ICC_ASGI1R_EL1 performs the same function as AArch32 System register ICC_ASGI1R.

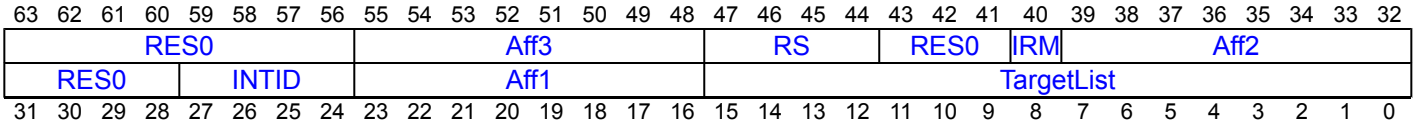
This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_ASGI1R_EL1 are UNDEFINED.

Under certain conditions a write to ICC_ASGI1R_EL1 can generate Group 0 interrupts, see 'Forwarding an SGI to a target PE' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_ASGI1R_EL1 is a 64-bit register.

Field descriptions



Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value ((RS * 16) + n).

When ICC_CTLR_EL1.RSS==0, RS is RES0.

When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a CONSTRAINED UNPREDICTABLE choice of:

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

IRM	Meaning
0b0	Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>.
0b1	Interrupts routed to all PEs in the system, excluding "self".

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

Note

If SRE is set only for EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Accessing ICC_ASGI1R_EL1

This register allows software executing in a Secure state to generate Non-secure Group 1 SGIs. It will also allow software executing in a Non-secure state to generate Secure Group 1 SGIs, if permitted by the settings of GICR_NSACR in the Redistributor corresponding to the target PE.

When GICD_CTLR.DS==0, Non-secure writes do not generate an interrupt for a target PE if not permitted by the GICR_NSACR register associated with the target PE. For more information, see 'Use of control registers for SGI forwarding'.

Note

Accesses at EL3 are treated as Secure regardless of the value of SCR_EL3.NS.

Accesses to this register use the following encodings in the System register encoding space:

```
MSR ICC_ASGI1R_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1011	0b110

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_ASGI1R_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_ASGI1R_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_ASGI1R_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_BPR0_EL1, Interrupt Controller Binary Point Register 0

The ICC_BPR0_EL1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

Configuration

AArch64 System register ICC_BPR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICC_BPR0[31:0].

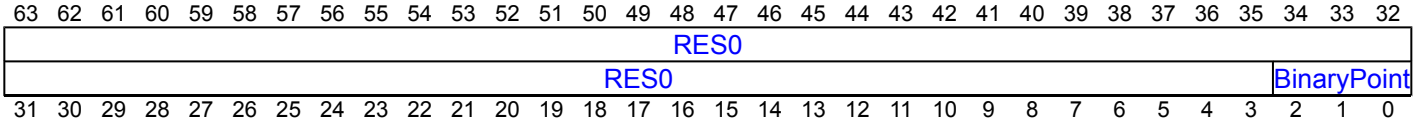
This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_BPR0_EL1 are UNDEFINED.

Virtual accesses to this register update ICH_VMCR_EL2.VBPR0.

Attributes

ICC_BPR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

Binary point value	Group priority field	Subpriority field	Field with binary point
0	[7:1]	[0]	ggggggg.s
1	[7:2]	[1:0]	gggggg.ss
2	[7:3]	[2:0]	ggggg.sss
3	[7:4]	[3:0]	gggg.ssss
4	[7:5]	[4:0]	ggg.sssss
5	[7:6]	[5:0]	gg.ssssss
6	[7]	[6:0]	g.sssssss
7	No preemption	[7:0]	.ssssssss

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_BPR0_EL1

The minimum binary point value is derived from the number of implemented priority bits. The number of priority bits is IMPLEMENTATION DEFINED, and reported by:

- ICC_CTLR_EL1.PRIbits
- ICC_CTLR_EL3.PRIbits An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

On a reset, the binary point field is UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_BPR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_BPR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_BPR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_BPR0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_BPR0_EL1;
```

```
MSR ICC_BPR0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_BPR0_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_BPR1_EL1, Interrupt Controller Binary Point Register 1

The ICC_BPR1_EL1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

Configuration

This register is banked between ICC_BPR1_EL1 and ICC_BPR1_EL1_S and ICC_BPR1_EL1_NS.

AArch64 System register ICC_BPR1_EL1 bits [31:0] (ICC_BPR1_EL1_S) are architecturally mapped to AArch32 System register ICC_BPR1[31:0] (ICC_BPR1_S).

AArch64 System register ICC_BPR1_EL1 bits [31:0] (ICC_BPR1_EL1_NS) are architecturally mapped to AArch32 System register ICC_BPR1[31:0] (ICC_BPR1_NS).

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_BPR1_EL1 are UNDEFINED.

Virtual accesses to this register update ICH_VMCR_EL2.VBPR1.

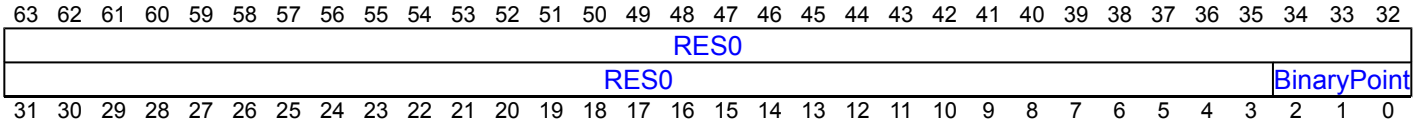
Attributes

ICC_BPR1_EL1 is a 64-bit register.

This register has the following instances:

- ICC_BPR1_EL1, when EL3 is not implemented.
- ICC_BPR1_EL1_S, when EL3 is implemented.
- ICC_BPR1_EL1_NS, when EL3 is implemented.

Field descriptions



Bits [63:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

If the GIC is configured to use separate binary point fields for Group 0 and Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. For more information about priorities, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The minimum value of the Non-secure copy of this register is the minimum value of ICC_BPR0_EL1 + 1. The minimum value of the Secure copy of this register is the minimum value of ICC_BPR0_EL1.

If EL3 is implemented and ICC_CTLR_EL3.CBPR_EL1S is 1:

- Accesses to this register from Secure EL2 access the state of ICC_BPR0_EL1.

- Accesses to this register from Secure EL1:
  - When SCR_EL3.EEL2 is 1 and HCR_EL2.IMO is 1, access the state of ICV_BPR1_EL1.
  - Otherwise, access the state of ICC_BPR0_EL1.

If EL3 is implemented and ICC_CTLR_EL3.CBPR_EL1NS is 1, Non-secure accesses to this register at EL1 and EL2 behave as follows, depending on the values of HCR_EL2.IMO and SCR_EL3.IRQ:

HCR_EL2.IMO	SCR_EL3.IRQ	Behavior
0b0	0b0	Non-secure EL1 and EL2 reads return ICC_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL1 and EL2 writes are ignored.
0b0	0b1	Non-secure EL1 and EL2 accesses trap to EL3.
0b1	0b0	Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 reads return ICC_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL2 writes are ignored.
0b1	0b1	Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 accesses trap to EL3.

If EL3 is not implemented and ICC_CTLR_EL1.CBPR is 1, Non-secure accesses to this register at EL1 and EL2 behave as follows, depending on the values of HCR_EL2.IMO:

HCR_EL2.IMO	Behavior
0b0	Non-secure EL1 and EL2 reads return ICC_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL1 and EL2 writes are ignored.
0b1	Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 reads return ICC_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL2 writes are ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_BPR1_EL1

On a reset, the binary point field is UNKNOWN.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_BPR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_BPR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_BPR1_EL1_S;
        else
            X[t, 64] = ICC_BPR1_EL1_NS;
    else
        X[t, 64] = ICC_BPR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_BPR1_EL1_S;
        else
            X[t, 64] = ICC_BPR1_EL1_NS;
    else
        X[t, 64] = ICC_BPR1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_BPR1_EL1_S;
        else
            X[t, 64] = ICC_BPR1_EL1_NS;
```

```
MSR ICC_BPR1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_BPR1_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t, 64];
        else
            ICC_BPR1_EL1_NS = X[t, 64];
    else
        ICC_BPR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t, 64];
        else
            ICC_BPR1_EL1_NS = X[t, 64];
    else
        ICC_BPR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t, 64];
        else
            ICC_BPR1_EL1_NS = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_CTLR_EL1, Interrupt Controller Control Register (EL1)

The ICC_CTLR_EL1 characteristics are:

Purpose

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Configuration

This register is banked between ICC_CTLR_EL1 and ICC_CTLR_EL1_S and ICC_CTLR_EL1_NS.

AArch64 System register ICC_CTLR_EL1 bits [31:0] (ICC_CTLR_EL1_S) are architecturally mapped to AArch32 System register ICC_CTLR[31:0] (ICC_CTLR_S).

AArch64 System register ICC_CTLR_EL1 bits [31:0] (ICC_CTLR_EL1_NS) are architecturally mapped to AArch32 System register ICC_CTLR[31:0] (ICC_CTLR_NS).

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_CTLR_EL1 are UNDEFINED.

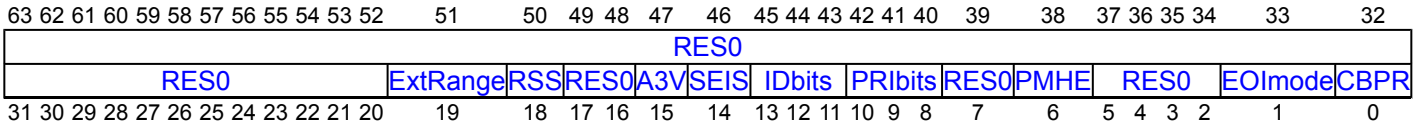
Attributes

ICC_CTLR_EL1 is a 64-bit register.

This register has the following instances:

- ICC_CTLR_EL1, when EL3 is not implemented.
- ICC_CTLR_EL1_S, when EL3 is implemented.
- ICC_CTLR_EL1_NS, when EL3 is implemented.

Field descriptions



Bits [63:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

ExtRange	Meaning
0b0	CPU interface does not support INTIDs in the range 1024..8191. • Behavior is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface. **Note** Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.
0b1	CPU interface supports INTIDs in the range 1024..8191 • All INTIDs in the range 1024..8191 are treated as requiring deactivation.

If EL3 is implemented, ICC_CTLR_EL1.ExtRange is an alias of ICC_CTLR_EL3.ExtRange.

RSS, bit [18]

Range Selector Support. Possible values are:

RSS	Meaning
0b0	Targeted SGIs with affinity level 0 values of 0 - 15 are supported.
0b1	Targeted SGIs with affinity level 0 values of 0 - 255 are supported.

This bit is read-only.

Bits [17:16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

A3V	Meaning
0b0	The CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.
0b1	The CPU interface logic supports nonzero values of Affinity 3 in SGI generation System registers.

If EL3 is implemented, this bit is an alias of ICC_CTLR_EL3.A3V.

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports local generation of SEIs:

SEIS	Meaning
0b0	The CPU interface logic does not support local generation of SEIs.
0b1	The CPU interface logic supports local generation of SEIs.

If EL3 is implemented, this bit is an alias of ICC_CTLR_EL3.SEIS.

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. The number of physical interrupt identifier bits supported:

IDbits	Meaning
0b000	16 bits.
0b001	24 bits.

All other values are reserved.

If EL3 is implemented, this field is an alias of ICC_CTLR_EL3.IDbits.

PRIbits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

Note

This field always returns the number of priority bits implemented, regardless of the Security state of the access or the value of GICD_CTLR.DS.

For physical accesses, this field determines the minimum value of ICC_BPR0_EL1.

If EL3 is implemented, physical accesses return the value from ICC_CTLR_EL3.PRIbits.

If EL3 is not implemented, physical accesses return the value from this field.

Bit [7]

Reserved, RES0.

PMHE, bit [6]

Priority Mask Hint Enable. Controls whether the priority mask register is used as a hint for interrupt distribution:

PMHE	Meaning
0b0	Disables use of ICC_PMR_EL1 as a hint for interrupt distribution.
0b1	Enables use of ICC_PMR_EL1 as a hint for interrupt distribution.

If EL3 is implemented, this bit is an alias of ICC_CTLR_EL3.PMHE. Whether this bit can be written as part of an access to this register depends on the value of GICD_CTLR.DS:

- If GICD_CTLR.DS == 0, this bit is read-only.
- If GICD_CTLR.DS == 1, this bit is read/write.

If EL3 is not implemented, it is IMPLEMENTATION DEFINED whether this bit is read-only or read/write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

Bits [5:2]

Reserved, RES0.

EOImode, bit [1]

EOI mode for the current Security state. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

EOImode	Meaning
0b0	ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE.
0b1	ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality.

The Secure ICC_CTLR_EL1.EOImode is an alias of ICC_CTLR_EL3.EOImode_EL1S.

The Non-secure ICC_CTLR_EL1.EOImode is an alias of ICC_CTLR_EL3.EOImode_EL1NS

CBPR, bit [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 interrupts:

CBPR	Meaning
0b0	ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts only. ICC_BPR1_EL1 determines the preemption group for Group 1 interrupts.
0b1	ICC_BPR0_EL1 determines the preemption group for both Group 0 and Group 1 interrupts.

If EL3 is implemented:

- This bit is an alias of ICC_CTLR_EL3.CBPR_EL1{S,NS} where S or NS corresponds to the current Security state.
- If GICD_CTLR.DS == 0, this bit is read-only.
- If GICD_CTLR.DS == 1, this bit is read/write.

If EL3 is not implemented, this bit is read/write.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_CTLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_CTLR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_CTLR_EL1;
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_CTLR_EL1;
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_CTLR_EL1_S;
        else
            X[t, 64] = ICC_CTLR_EL1_NS;
    else
        X[t, 64] = ICC_CTLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_CTLR_EL1_S;
        else
            X[t, 64] = ICC_CTLR_EL1_NS;
    else
        X[t, 64] = ICC_CTLR_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_CTLR_EL1_S;
        else
            X[t, 64] = ICC_CTLR_EL1_NS;
```

```
MSR ICC_CTLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_CTLR_EL1 = X[t, 64];
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_CTLR_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t, 64];
        else
            ICC_CTLR_EL1_NS = X[t, 64];
    else
        ICC_CTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t, 64];
        else
            ICC_CTLR_EL1_NS = X[t, 64];
    else
        ICC_CTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t, 64];
        else
            ICC_CTLR_EL1_NS = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_CTLR_EL3, Interrupt Controller Control Register (EL3)

The ICC_CTLR_EL3 characteristics are:

Purpose

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

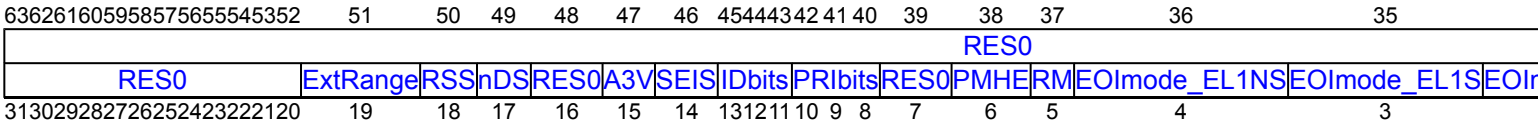
Configuration

This register is present only when GICv3 is implemented, EL3 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_CTLR_EL3 are UNDEFINED.

Attributes

ICC_CTLR_EL3 is a 64-bit register.

Field descriptions



Bits [63:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

ExtRange	Meaning
0b0	CPU interface does not support INTIDs in the range 1024..8191. • Behavior is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface. **Note** Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.
0b1	CPU interface supports INTIDs in the range 1024..8191 • All INTIDs in the range 1024..8191 are treated as requiring deactivation.

RSS, bit [18]

Range Selector Support.

RSS	Meaning
0b0	Targeted SGIs with affinity level 0 values of 0-15 are supported.
0b1	Targeted SGIs with affinity level 0 values of 0-255 are supported.

This bit is read-only.

nDS, bit [17]

Disable Security not supported. Read-only and writes are ignored.

nDS	Meaning
0b0	The CPU interface logic supports disabling of security.
0b1	The CPU interface logic does not support disabling of security, and requires that security is not disabled.

When a PE implements FEAT_RME and FEAT_SEL2, this field is RAO/WI.

Bit [16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored.

A3V	Meaning
0b0	The CPU interface logic does not support nonzero values of the Aff3 field in SGI generation System registers.
0b1	The CPU interface logic supports nonzero values of the Aff3 field in SGI generation System registers.

If EL3 is present, ICC_CTLR_EL1.A3V is an alias of ICC_CTLR_EL3.A3V

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports generation of SEIs:

SEIS	Meaning
0b0	The CPU interface logic does not support generation of SEIs.
0b1	The CPU interface logic supports generation of SEIs.

If EL3 is present, ICC_CTLR_EL1.SEIS is an alias of ICC_CTLR_EL3.SEIS

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. Indicates the number of physical interrupt identifier bits supported.

IDbits	Meaning
0b000	16 bits.
0b001	24 bits.

All other values are reserved.

If EL3 is present, ICC_CTLR_EL1.IDbits is an alias of ICC_CTLR_EL3.IDbits

PRIbits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

Note

This field always returns the number of priority bits implemented, regardless of the value of SCR_EL3.NS or the value of GICD_CTLR.DS.

The division between group priority and subpriority is defined in the binary point registers ICC_BPR0_EL1 and ICC_BPR1_EL1.

This field determines the minimum value of ICC_BPR0_EL1.

Bit [7]

Reserved, RES0.

PMHE, bit [6]

Priority Mask Hint Enable.

PMHE	Meaning
0b0	Disables use of the priority mask register as a hint for interrupt distribution.
0b1	Enables use of the priority mask register as a hint for interrupt distribution.

Software must write ICC_PMR_EL1 to 0xFF before clearing this field to 0.

- An implementation might choose to make this field RAO/WI if priority-based routing is always used
- An implementation might choose to make this field RAZ/WI if priority-based routing is never used

If EL3 is present, ICC_CTLR_EL1.PMHE is an alias of ICC_CTLR_EL3.PMHE.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

RM, bit [5]

Routing Modifier. This bit controls whether EL3 can acknowledge, or observe as the Highest Priority Pending Interrupt, Secure Group 0 and Non-secure Group 1 interrupts.

RM	Meaning
0b0	Secure Group 0 and Non-secure Group 1 interrupts can be acknowledged and observed as the highest priority interrupt at EL3.
0b1	Secure Group 0 and Non-secure Group 1 interrupts cannot be acknowledged and observed as the highest priority interrupt at EL3. Secure Group 0 interrupts return a special INTID value of 1020. This affects accesses to ICC_IAR0_EL1 and ICC_HPPIR0_EL1. Non-secure Group 1 interrupts return a special INTID value of 1021. This affects accesses to ICC_IAR1_EL1 and ICC_HPPIR1_EL1.

Note

The Routing Modifier bit is supported in AArch64 only. In systems without EL3 the behavior is as if the value is 0. Software must ensure this bit is 0 when the Secure copy of ICC_SRE_EL1.SRE is 1, otherwise system behavior is UNPREDICTABLE. In systems without EL3 or where the Secure copy of ICC_SRE_EL1.SRE is RAO/WI, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL1NS, bit [4]

EOI mode for interrupts handled at Non-secure EL1 and EL2. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOImode_EL1NS	Meaning
0b0	ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE.
0b1	ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality.

If EL3 is present, ICC_CTLR_EL1(NS).EOImode is an alias of ICC_CTLR_EL3.EOImode_EL1NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL1S, bit [3]

EOI mode for interrupts handled at Secure EL1 and EL2. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOImode_EL1S	Meaning
0b0	ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE.
0b1	ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality.

If EL3 is present, ICC_CTLR_EL1(S).EOImode is an alias of ICC_CTLR_EL3.EOImode_EL1S.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL3, bit [2]

EOI mode for interrupts handled at EL3. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOImode_EL3	Meaning
0b0	ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE.
0b1	ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR_EL1NS, bit [1]

Common Binary Point Register, EL1 Non-secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Non-secure interrupts at EL1 and EL2.

CBPR_EL1NS	Meaning
0b0	ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts only. ICC_BPR1_EL1 determines the preemption group for Non-secure Group 1 interrupts.
0b1	ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts and Non-secure Group 1 interrupts. Non-secure accesses to GICC_BPR and ICC_BPR1_EL1 access the state of ICC_BPR0_EL1.

If EL3 is present, ICC_CTLR_EL1(NS).CBPR is an alias of ICC_CTLR_EL3.CBPR_EL1NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR_EL1S, bit [0]

Common Binary Point Register, EL1 Secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Secure interrupts at EL1 and EL2.

CBPR_EL1S	Meaning
0b0	ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts only. ICC_BPR1_EL1 determines the preemption group for Secure Group 1 interrupts.
0b1	ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts and Secure Group 1 interrupts. Secure EL1 and Secure EL2 accesses to ICC_BPR1_EL1 access the state of ICC_BPR0_EL1.

If EL3 is present, ICC_CTLR_EL1(S).CBPR is an alias of ICC_CTLR_EL3.CBPR_EL1S.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_CTLR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_CTLR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_CTLR_EL3;
```

```
MSR ICC_CTLR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_CTLR_EL3 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_DIR_EL1, Interrupt Controller Deactivate Interrupt Register

The ICC_DIR_EL1 characteristics are:

Purpose

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified interrupt.

Configuration

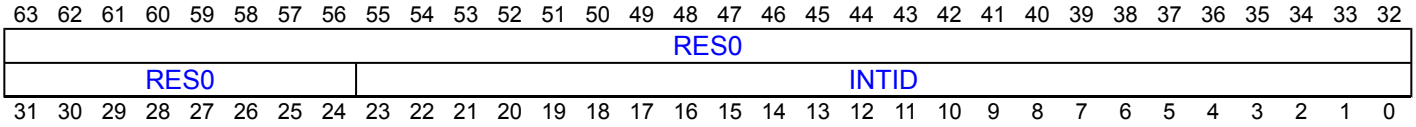
AArch64 System register ICC_DIR_EL1 bits [31:0] performs the same function as AArch32 System register ICC_DIR[31:0].

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_DIR_EL1 are UNDEFINED.

Attributes

ICC_DIR_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_DIR_EL1

There are two cases when writing to ICC_DIR_EL1 that were UNPREDICTABLE for a corresponding GICv2 write to GICC_DIR:

- When EOImode == 0. GICv3 implementations must ignore such writes. In systems supporting system error generation, an implementation might generate an SEI.
- When EOImode == 1 but no EOI has been issued. The interrupt will be deactivated by the Distributor, however the active priority in the CPU interface for the interrupt will remain set (because no EOI was issued).

Accesses to this register use the following encodings in the System register encoding space:

```
MSR ICC_DIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1011	0b001

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TDIR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_DIR_EL1 = X[t, 64];
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_DIR_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_EOIR0_EL1, Interrupt Controller End Of Interrupt Register 0

The ICC_EOIR0_EL1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 0 interrupt.

Configuration

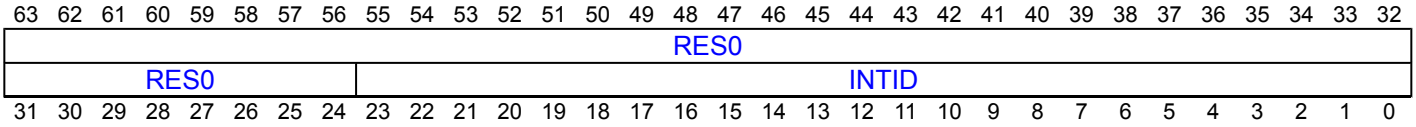
AArch64 System register ICC_EOIR0_EL1 bits [31:0] performs the same function as AArch32 System register ICC_EOIR0[31:0].

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_EOIR0_EL1 are UNDEFINED.

Attributes

ICC_EOIR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding ICC_IAR0_EL1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to ICC_DIR_EL1 to deactivate the interrupt.

The EOImode bit for the current Exception level and Security state is determined as follows:

- If EL3 is not implemented, the appropriate bit is ICC_CTLR_EL1.EOIMode.
- If EL3 is implemented and the software is executing at EL3, the appropriate bit is ICC_CTLR_EL3.EOImode_EL3.
- If EL3 is implemented and the software is not executing at EL3, the bit depends on the current Security state:
  - If the software is executing in Secure state, the bit is ICC_CTLR_EL3.EOImode_EL1S.
  - If the software is executing in Non-secure state, the bit is ICC_CTLR_EL3.EOImode_EL1NS.

Accessing ICC_EOIR0_EL1

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICC_IAR0_EL1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accesses to this register use the following encodings in the System register encoding space:

```
MSR ICC_EOIR0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b001

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_EOIR0_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_EOIR1_EL1, Interrupt Controller End Of Interrupt Register 1

The ICC_EOIR1_EL1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 1 interrupt.

Configuration

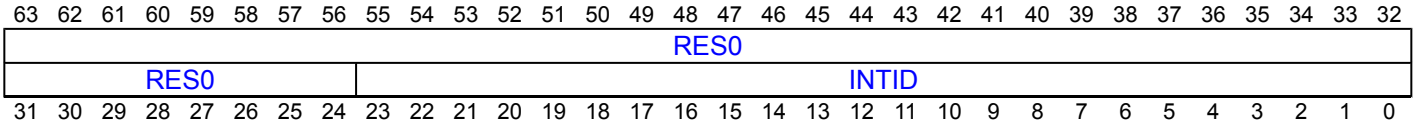
AArch64 System register ICC_EOIR1_EL1 bits [31:0] performs the same function as AArch32 System register ICC_EOIR1[31:0].

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_EOIR1_EL1 are UNDEFINED.

Attributes

ICC_EOIR1_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding ICC_IAR1_EL1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to ICC_DIR_EL1 to deactivate the interrupt.

The EOImode bit for the current Exception level and Security state is determined as follows:

- If EL3 is not implemented, the appropriate bit is ICC_CTLR_EL1.EOIMode.
- If EL3 is implemented and the software is executing at EL3, the appropriate bit is ICC_CTLR_EL3.EOImode_EL3.
- If EL3 is implemented and the software is not executing at EL3, the bit depends on the current Security state:
  - If the software is executing in Secure state, the bit is ICC_CTLR_EL3.EOImode_EL1S.
  - If the software is executing in Non-secure state, the bit is ICC_CTLR_EL3.EOImode_EL1NS.

Accessing ICC_EOIR1_EL1

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICC_IAR1_EL1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accesses to this register use the following encodings in the System register encoding space:

```
MSR ICC_EOIR1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_EOIR1_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_HPPIR0_EL1, Interrupt Controller Highest Priority Pending Interrupt Register 0

The ICC_HPPIR0_EL1 characteristics are:

Purpose

Indicates the highest priority pending Group 0 interrupt on the CPU interface.

Configuration

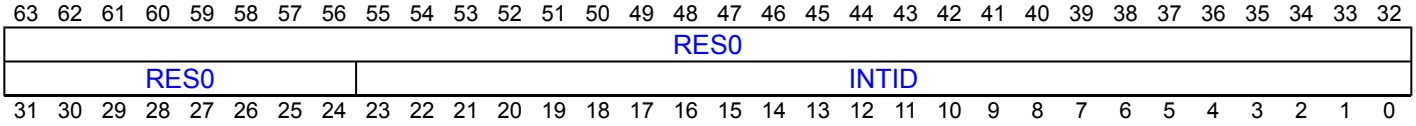
AArch64 System register ICC_HPPIR0_EL1 bits [31:0] performs the same function as AArch32 System register ICC_HPPIR0[31:0].

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_HPPIR0_EL1 are UNDEFINED.

Attributes

ICC_HPPIR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_HPPIR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_HPPIR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b010

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_HPPIR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_HPPIR1_EL1, Interrupt Controller Highest Priority Pending Interrupt Register 1

The ICC_HPPIR1_EL1 characteristics are:

Purpose

Indicates the highest priority pending Group 1 interrupt on the CPU interface.

Configuration

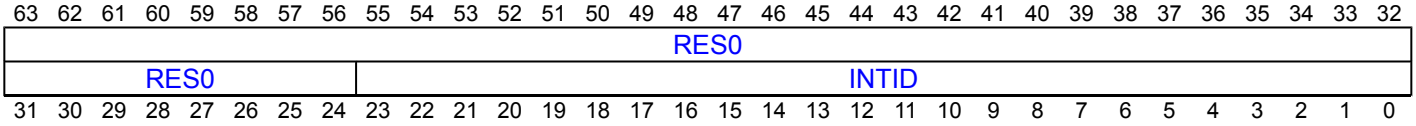
AArch64 System register ICC_HPPIR1_EL1 bits [31:0] performs the same function as AArch32 System register ICC_HPPIR1[31:0].

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_HPPIR1_EL1 are UNDEFINED.

Attributes

ICC_HPPIR1_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_HPPIR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_HPPIR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_HPPIR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IAR0_EL1, Interrupt Controller Interrupt Acknowledge Register 0

The ICC_IAR0_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled Group 0 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch64 System register ICC_IAR0_EL1 bits [31:0] performs the same function as AArch32 System register ICC_IAR0[31:0].

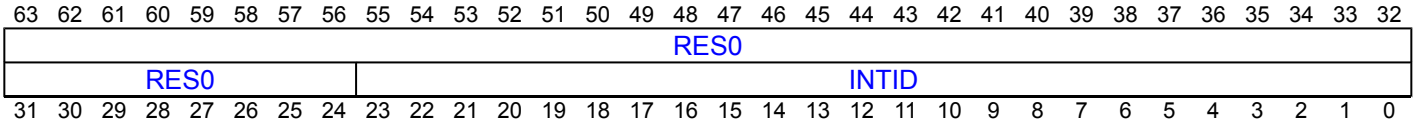
This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_IAR0_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronizing when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers'.

Attributes

ICC_IAR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_IAR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_IAR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_IAR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IAR1_EL1, Interrupt Controller Interrupt Acknowledge Register 1

The ICC_IAR1_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch64 System register ICC_IAR1_EL1 bits [31:0] performs the same function as AArch32 System register ICC_IAR1[31:0].

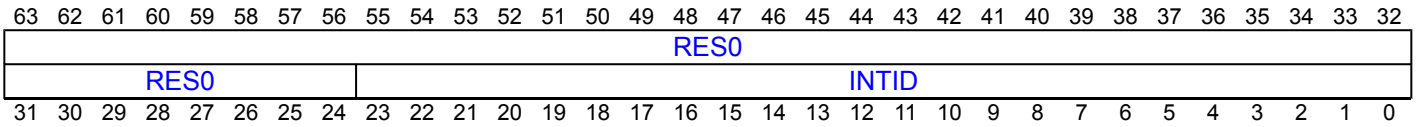
This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_IAR1_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronizing when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_IAR1_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_IAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_IAR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_IAR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IGRPEN0_EL1, Interrupt Controller Interrupt Group 0 Enable Register

The ICC_IGRPEN0_EL1 characteristics are:

Purpose

Controls whether Group 0 interrupts are enabled or not.

Configuration

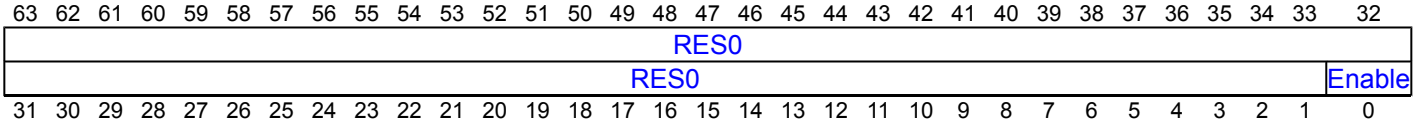
AArch64 System register ICC_IGRPEN0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICC_IGRPEN0[31:0].

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_IGRPEN0_EL1 are UNDEFINED.

Attributes

ICC_IGRPEN0_EL1 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

Enable, bit [0]

Enables Group 0 interrupts.

Enable	Meaning
0b0	Group 0 interrupts are disabled.
0b1	Group 0 interrupts are enabled.

Virtual accesses to this register update ICH_VMCR_EL2.VENG0.

If the highest priority pending interrupt for that PE is a Group 0 interrupt using 1 of N model, then the interrupt will be targeted to another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICC_IGRPEN0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_IGRPEN0_EL1
```

op0	op1	CRn	CRm	op2

0b11	0b000	0b1100	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_IGRPEN0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IGRPEN0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IGRPEN0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IGRPEN0_EL1;
```

```
MSR ICC_IGRPEN0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_IGRPEN0_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IGRPEN1_EL1, Interrupt Controller Interrupt Group 1 Enable Register

The ICC_IGRPEN1_EL1 characteristics are:

Purpose

Controls whether Group 1 interrupts are enabled for the current Security state.

Configuration

This register is banked between ICC_IGRPEN1_EL1 and ICC_IGRPEN1_EL1_S and ICC_IGRPEN1_EL1_NS.

AArch64 System register ICC_IGRPEN1_EL1 bits [31:0] (ICC_IGRPEN1_EL1_S) are architecturally mapped to AArch32 System register ICC_IGRPEN1[31:0] (ICC_IGRPEN1_S).

AArch64 System register ICC_IGRPEN1_EL1 bits [31:0] (ICC_IGRPEN1_EL1_NS) are architecturally mapped to AArch32 System register ICC_IGRPEN1[31:0] (ICC_IGRPEN1_NS).

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_IGRPEN1_EL1 are UNDEFINED.

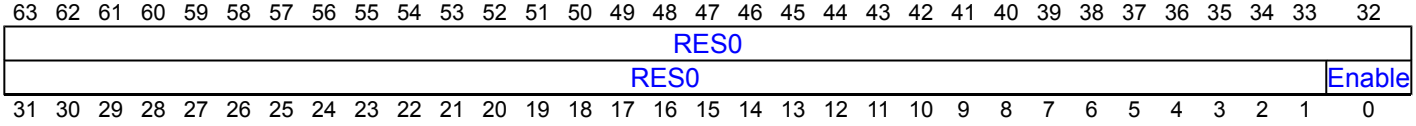
Attributes

ICC_IGRPEN1_EL1 is a 64-bit register.

This register has the following instances:

- ICC_IGRPEN1_EL1, when EL3 is not implemented.
- ICC_IGRPEN1_EL1_S, when EL3 is implemented.
- ICC_IGRPEN1_EL1_NS, when EL3 is implemented.

Field descriptions



Bits [63:1]

Reserved, RES0.

Enable, bit [0]

Enables Group 1 interrupts for the current Security state.

Enable	Meaning
0b0	Group 1 interrupts are disabled for the current Security state.
0b1	Group 1 interrupts are enabled for the current Security state.

Virtual accesses to this register update ICH_VMCR_EL2.VENG1.

If EL3 is present:

- The Secure ICC_IGRPEN1_EL1.Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1S bit.
- The Non-secure ICC_IGRPEN1_EL1.Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1NS bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing ICC_IGRPEN1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_IGRPEN1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_IGRPEN1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_IGRPEN1_EL1_S;
        else
            X[t, 64] = ICC_IGRPEN1_EL1_NS;
    else
        X[t, 64] = ICC_IGRPEN1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_IGRPEN1_EL1_S;
        else
            X[t, 64] = ICC_IGRPEN1_EL1_NS;
    else
        X[t, 64] = ICC_IGRPEN1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_IGRPEN1_EL1_S;
        else
            X[t, 64] = ICC_IGRPEN1_EL1_NS;
```

```
MSR ICC_IGRPEN1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_IGRPEN1_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t, 64];
        else
            ICC_IGRPEN1_EL1_NS = X[t, 64];
    else
        ICC_IGRPEN1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t, 64];
        else
            ICC_IGRPEN1_EL1_NS = X[t, 64];
    else
        ICC_IGRPEN1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t, 64];
        else
            ICC_IGRPEN1_EL1_NS = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IGRPEN1_EL3, Interrupt Controller Interrupt Group 1 Enable Register (EL3)

The ICC_IGRPEN1_EL3 characteristics are:

Purpose

Controls whether Group 1 interrupts are enabled or not.

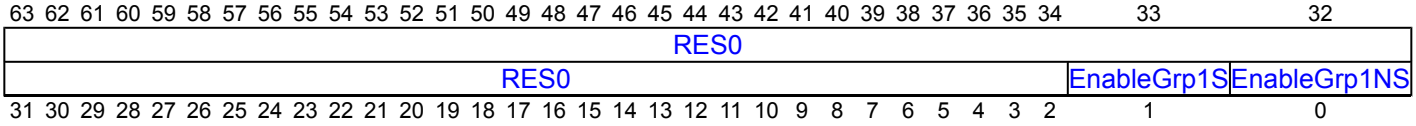
Configuration

This register is present only when GICv3 is implemented, EL3 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_IGRPEN1_EL3 are UNDEFINED.

Attributes

ICC_IGRPEN1_EL3 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

EnableGrp1S, bit [1]

Enables Group 1 interrupts for the Secure state.

EnableGrp1S	Meaning
0b0	Secure Group 1 interrupts are disabled.
0b1	Secure Group 1 interrupts are enabled.

The Secure ICC_IGRPEN1_EL1.Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1S bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

EnableGrp1NS, bit [0]

Enables Group 1 interrupts for the Non-secure state.

EnableGrp1NS	Meaning
0b0	Non-secure Group 1 interrupts are disabled.
0b1	Non-secure Group 1 interrupts are enabled.

The Non-secure ICC_IGRPEN1_EL1.Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1NS bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing ICC_IGRPEN1_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_IGRPEN1_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IGRPEN1_EL3;
```

```
MSR ICC_IGRPEN1_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN1_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_NMIAR1_EL1, Interrupt Controller Non-maskable Interrupt Acknowledge Register 1

The ICC_NMIAR1_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled Group 1 non-maskable interrupt. This read acts as an acknowledge for the interrupt.

Configuration

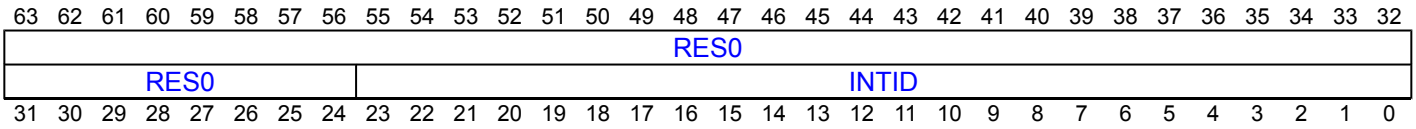
This register is present only when FEAT_GICv3_NMI is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_NMIAR1_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronizing when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_NMIAR1_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt has the Non-maskable property and is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_NMIAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_NMIAR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1001	0b101

```
if !(IsFeatureImplemented(FEAT_GICv3_NMI) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if SCTLR_EL1.NMI == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_NMIAR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_NMIAR1_EL1;
elsif PSTATE.EL == EL2 then
    if SCTLR_EL2.NMI == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_NMIAR1_EL1;
elsif PSTATE.EL == EL3 then
    if SCTLR_EL3.NMI == '0' then
        UNDEFINED;
    elsif ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_NMIAR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_PMR_EL1, Interrupt Controller Interrupt Priority Mask Register

The ICC_PMR_EL1 characteristics are:

Purpose

Provides an interrupt priority filter. Only interrupts with a higher priority than the value in this register are signaled to the PE.

Writes to this register must be high performance and must ensure that no interrupt of lower priority than the written value occurs after the write, without requiring an ISB or an exception boundary.

Configuration

AArch64 System register ICC_PMR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICC_PMR[31:0].

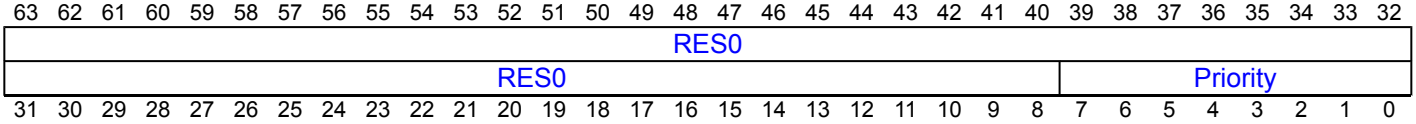
This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_PMR_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronising. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_PMR_EL1 is a 64-bit register.

Field descriptions



Bits [63:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the CPU interface. If the priority of an interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

The possible priority field values are as follows:

Implemented priority bits	Possible priority field values	Number of priority levels
[7:0]	0x00-0xFF (0-255), all values	256
[7:1]	0x00-0xFE (0-254), even values only	128
[7:2]	0x00-0xFC (0-252), in steps of 4	64
[7:3]	0x00-0xF8 (0-248), in steps of 8	32
[7:4]	0x00-0xF0 (0-240), in steps of 16	16

Unimplemented priority bits are RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00000000'.

Accessing ICC_PMR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_PMR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_PMR_EL1;
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PMR_EL1;
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_PMR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_PMR_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_PMR_EL1;
```

MSR ICC_PMR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_PMR_EL1 = X[t, 64];
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PMR_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_RPR_EL1, Interrupt Controller Running Priority Register

The ICC_RPR_EL1 characteristics are:

Purpose

Indicates the Running priority of the CPU interface.

Configuration

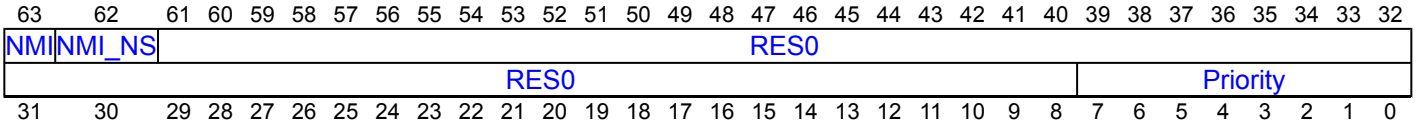
AArch64 System register ICC_RPR_EL1 bits [31:0] performs the same function as AArch32 System register ICC_RPR[31:0].

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_RPR_EL1 are UNDEFINED.

Attributes

ICC_RPR_EL1 is a 64-bit register.

Field descriptions



NMI, bit [63]

When FEAT_GICv3_NMI is implemented:

Indicates whether the running priority is from a NMI.

NMI	Meaning
0b0	When GICD_CTLR.DS==1, there are no Active NMIs, or all Active NMIs have undergone priority drop. When GICD_CTLR.DS==0: • For Non-secure and Realm reads, there are no Active Non-secure Group 1 NMIs, or all Active Non-secure Group 1 NMIs have undergone priority drop. • For Secure and Root reads, there are no Active Secure Group 1 NMIs, or all Active Secure Group 1 NMIs have undergone priority drop.
0b1	When GICD_CTLR.DS==1, there is an Active NMI. When GICD_CTLR.DS==0: • For Non-secure and Realm reads, there is an Active Non-secure Group 1 NMI. • For Secure and Root reads, there is an Active Secure Group 1 NMI.

Otherwise:

Reserved, RES0.

NMI_NS, bit [62]

When FEAT_GICv3_NMI is implemented and EL3 is implemented:

Indicates whether the running priority is from a Non-secure Group 1 NMI.

NMI_NS	Meaning
0b0	There are no Active Non-secure Group 1 NMIs, or all Active Non-secure Group 1 NMIs have undergone priority drop.
0b1	There is an Active Non-secure Group 1 NMI which has not undergone priority drop.

For Non-secure and Realm accesses, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [61:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the CPU interface. This is the group priority of the current active interrupt.

The group priority of a Secure NMI, or NMI when GICD_CTLR.DS is 1, is 0x00. The group priority of a Non-secure NMI is 0x80, saturated to 0x00 for Non-secure reads.

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

Note

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

Accessing ICC_RPR_EL1

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_RPR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_RPR_EL1;
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_RPR_EL1;
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_RPR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_RPR_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_RPR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SGI0R_EL1, Interrupt Controller Software Generated Interrupt Group 0 Register

The ICC_SGI0R_EL1 characteristics are:

Purpose

Generates Secure Group 0 SGIs.

Configuration

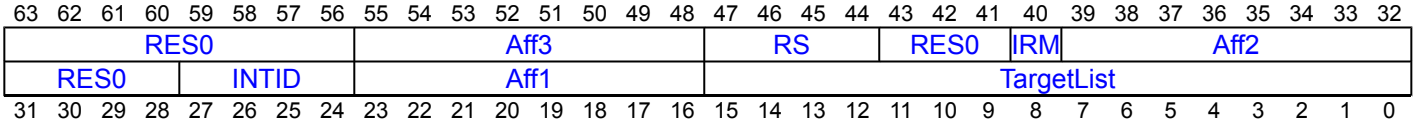
AArch64 System register ICC_SGI0R_EL1 performs the same function as AArch32 System register ICC_SGI0R.

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_SGI0R_EL1 are UNDEFINED.

Attributes

ICC_SGI0R_EL1 is a 64-bit register.

Field descriptions



Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value ((RS * 16) + n).

When ICC_CTLR_EL1.RSS==0, RS is RES0.

When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a CONSTRAINED UNPREDICTABLE choice of:

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

IRM	Meaning
0b0	Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>.
0b1	Interrupts routed to all PEs in the system, excluding "self".

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

Note

If SRE is set only for EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Accessing ICC_SGI0R_EL1

This register allows software executing in a Secure state to generate Group 0 SGIs. It will also allow software executing in a Non-secure state to generate Group 0 SGIs, if permitted by the settings of GICR_NSACR in the Redistributor corresponding to the target PE.

When GICD_CTLR.DS==0, Non-secure writes do not generate an interrupt for a target PE if not permitted by the GICR_NSACR register associated with the target PE. For more information, see 'Use of control registers for SGI forwarding' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Accesses at EL3 are treated as Secure regardless of the value of SCR_EL3.NS.

Accesses to this register use the following encodings in the System register encoding space:

```
MSR ICC_SGI0R_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1011	0b111

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SGI0R_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SGI0R_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SGI0R_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SGI1R_EL1, Interrupt Controller Software Generated Interrupt Group 1 Register

The ICC_SGI1R_EL1 characteristics are:

Purpose

Generates Group 1 SGIs for the current Security state.

Configuration

AArch64 System register ICC_SGI1R_EL1 performs the same function as AArch32 System register ICC_SGI1R.

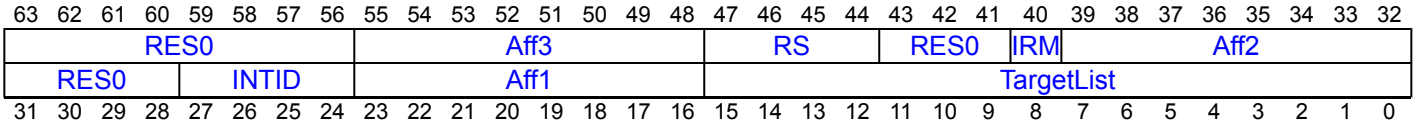
This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_SGI1R_EL1 are UNDEFINED.

Under certain conditions a write to ICC_SGI1R_EL1 can generate Group 0 interrupts, see 'Forwarding an SGI to a target PE' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_SGI1R_EL1 is a 64-bit register.

Field descriptions



Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value ((RS * 16) + n).

When ICC_CTLR_EL1.RSS==0, RS is RES0.

When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a CONSTRAINED UNPREDICTABLE choice of:

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

IRM	Meaning
0b0	Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>.
0b1	Interrupts routed to all PEs in the system, excluding "self".

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

Note

If SRE is set only for EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Accessing ICC_SGI1R_EL1

Note

Accesses at EL3 are treated as Secure regardless of the value of SCR_EL3.NS.

Accesses to this register use the following encodings in the System register encoding space:

```
MSR ICC_SGI1R_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1011	0b101

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SGI1R_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SGI1R_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SGI1R_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SRE_EL1, Interrupt Controller System Register Enable Register (EL1)

The ICC_SRE_EL1 characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL1.

Configuration

This register is banked between ICC_SRE_EL1 and ICC_SRE_EL1_S and ICC_SRE_EL1_NS.

AArch64 System register ICC_SRE_EL1 bits [31:0] (ICC_SRE_EL1_S) are architecturally mapped to AArch32 System register ICC_SRE[31:0] (ICC_SRE_S).

AArch64 System register ICC_SRE_EL1 bits [31:0] (ICC_SRE_EL1_NS) are architecturally mapped to AArch32 System register ICC_SRE[31:0] (ICC_SRE_NS).

AArch64 System register ICC_SRE_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICC_SRE.

This register is present only when GICv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_SRE_EL1 are UNDEFINED.

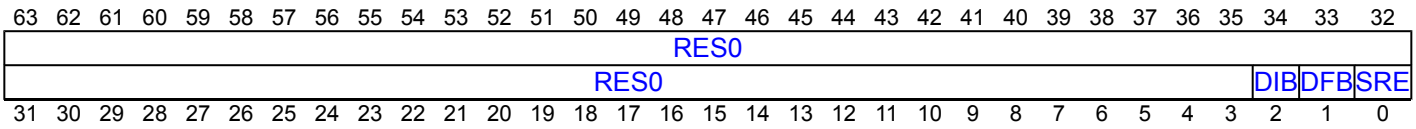
Attributes

ICC_SRE_EL1 is a 64-bit register.

This register has the following instances:

- ICC_SRE_EL1, when EL3 is not implemented.
- ICC_SRE_EL1_S, when EL3 is implemented.
- ICC_SRE_EL1_NS, when EL3 is implemented.

Field descriptions



Bits [63:3]

Reserved, RES0.

DIB, bit [2]

Disable IRQ bypass.

DIB	Meaning
0b0	IRQ bypass enabled.
0b1	IRQ bypass disabled.

If EL3 is implemented and GICD_CTLR.DS == 0, this field is a read-only alias of ICC_SRE_EL3.DIB.

If EL3 is implemented and GICD_CTLR.DS == 1, and EL2 is not implemented, this field is a read/write alias of ICC_SRE_EL3.DIB.

If EL3 is not implemented and EL2 is implemented, this field is a read-only alias of ICC_SRE_EL2.DIB.

If GICD_CTLR.DS == 1 and EL2 is implemented, this field is a read-only alias of ICC_SRE_EL2.DIB.

In systems that do not support IRQ bypass, this field is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

DFB, bit [1]

Disable FIQ bypass.

DFB	Meaning
0b0	FIQ bypass enabled.
0b1	FIQ bypass disabled.

If EL3 is implemented and GICD_CTLR.DS == 0, this field is a read-only alias of ICC_SRE_EL3.DFB.

If EL3 is implemented and GICD_CTLR.DS == 1, and EL2 is not implemented, this field is a read/write alias of ICC_SRE_EL3.DFB.

If EL3 is not implemented and EL2 is implemented, this field is a read-only alias of ICC_SRE_EL2.DFB.

If GICD_CTLR.DS == 1 and EL2 is implemented, this field is a read-only alias of ICC_SRE_EL2.DFB.

In systems that do not support FIQ bypass, this field is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

SRE, bit [0]

System Register Enable.

SRE	Meaning
0b0	The memory-mapped interface must be used. Access at EL1 to any ICC_* System register other than ICC_SRE_EL1 is trapped to EL1.
0b1	The System register interface for the current Security state is enabled.

If software changes this bit from 1 to 0 in the Secure instance of this register, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and ICC_SRE_EL3.SRE==0 the Secure copy of this bit is RAZ/WI. If ICC_SRE_EL3.SRE is changed from zero to one, the Secure copy of this bit becomes UNKNOWN.

If EL2 is implemented and ICC_SRE_EL2.SRE==0 the Non-secure copy of this bit is RAZ/WI. If ICC_SRE_EL2.SRE is changed from zero to one, the Non-secure copy of this bit becomes UNKNOWN.

If EL3 is implemented and ICC_SRE_EL3.SRE==0 the Non-secure copy of this bit is RAZ/WI. If ICC_SRE_EL3.SRE is changed from zero to one, the Non-secure copy of this bit becomes UNKNOWN.

If Realm Management Extension is implemented, this field is RAO/WI.

GICv3 implementations that do not require GICv2 compatibility might choose to make this bit RAO/WI. The following options are supported:

- The Non-secure copy of ICC_SRE_EL1.SRE can be RAO/WI if ICC_SRE_EL2.SRE is also RAO/WI. This means all Non-secure software, including VMs using only virtual interrupts, must access the GIC using System registers.
- The Secure copy of ICC_SRE_EL1.SRE can be RAO/WI if ICC_SRE_EL3.SRE and ICC_SRE_EL2.SRE are also RAO/WI. This means that all Secure software must access the GIC using System registers and all Non-secure accesses to registers for physical interrupts must use System registers.

Note

A VM using only virtual interrupts might still use memory-mapped access if the Non-secure copy of ICC_SRE_EL1.SRE is not RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICC_SRE_EL1

Execution with ICC_SRE_EL1.SRE set to 0 might make some System registers UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_SRE_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif EL2Enabled() && ICC_SRE_EL2.Enable == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_SRE_EL1_S;
        else
            X[t, 64] = ICC_SRE_EL1_NS;
    else
        X[t, 64] = ICC_SRE_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_SRE_EL1_S;
        else
            X[t, 64] = ICC_SRE_EL1_NS;
    else
        X[t, 64] = ICC_SRE_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        X[t, 64] = ICC_SRE_EL1_S;
    else
        X[t, 64] = ICC_SRE_EL1_NS;
```

```
MSR ICC_SRE_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif EL2Enabled() && ICC_SRE_EL2.Enable == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_SRE_EL1_S = X[t, 64];
        else
            ICC_SRE_EL1_NS = X[t, 64];
    else
        ICC_SRE_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_SRE_EL1_S = X[t, 64];
        else
            ICC_SRE_EL1_NS = X[t, 64];
    else
        ICC_SRE_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        ICC_SRE_EL1_S = X[t, 64];
    else
        ICC_SRE_EL1_NS = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SRE_EL2, Interrupt Controller System Register Enable Register (EL2)

The ICC_SRE_EL2 characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL2.

Configuration

AArch64 System register ICC_SRE_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICC_HSRE[31:0].

This register is present only when GICv3 is implemented, (EL2 is implemented or EL3 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_SRE_EL2 are UNDEFINED.

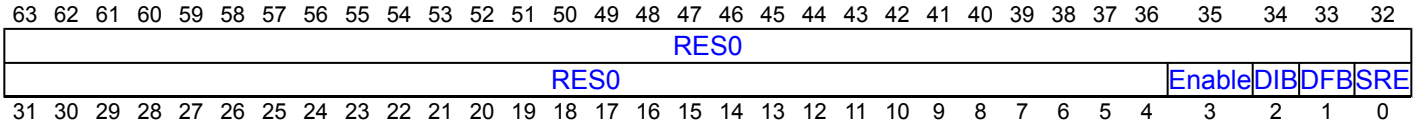
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICC_SRE_EL2 is a 64-bit register.

Field descriptions



Bits [63:4]

Reserved, RES0.

Enable, bit [3]

Enable. Enables lower Exception level access to ICC_SRE_EL1.

Enable	Meaning
0b0	When EL2 is implemented and enabled in the current Security state, EL1 accesses to ICC_SRE_EL1 trap to EL2.
0b1	EL1 accesses to ICC_SRE_EL1 do not trap to EL2.

If ICC_SRE_EL2.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_SRE_EL2.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIB, bit [2]

Disable IRQ bypass.

DIB	Meaning
0b0	IRQ bypass enabled.
0b1	IRQ bypass disabled.

If EL3 is implemented and GICD_CTLR.DS is 0, this field is a read-only alias of ICC_SRE_EL3.DIB.

If EL3 is implemented and GICD_CTLR.DS is 1, this field is a read/write alias of ICC_SRE_EL3.DIB.

In systems that do not support IRQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

DFB, bit [1]

Disable FIQ bypass.

DFB	Meaning
0b0	FIQ bypass enabled.
0b1	FIQ bypass disabled.

If EL3 is implemented and GICD_CTLR.DS is 0, this field is a read-only alias of ICC_SRE_EL3.DFB.

If EL3 is implemented and GICD_CTLR.DS is 1, this field is a read/write alias of ICC_SRE_EL3.DFB.

In systems that do not support FIQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

SRE, bit [0]

System Register Enable.

SRE	Meaning
0b0	The memory-mapped interface must be used. Access at EL2 to any ICH_* or ICC_* register other than ICC_SRE_EL1 or ICC_SRE_EL2, is trapped to EL2.
0b1	The System register interface to the ICH_* registers and the EL1 and EL2 ICC_* registers is enabled for EL2.

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and ICC_SRE_EL3.SRE==0 this bit is RAZ/WI. If ICC_SRE_EL3.SRE is changed from zero to one, this bit becomes UNKNOWN.

If Realm Management Extension is implemented, this field is RAO/WI.

FEAT_GICv3 implementations that do not require GICv2 compatibility might choose to make this bit RAO/WI, but this is only allowed if ICC_SRE_EL3.SRE is also RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICC_SRE_EL2

Execution with ICC_SRE_EL2.SRE set to 0 might make some System registers UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_SRE_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1001	0b101

```
if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_SRE_EL2;
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        X[t, 64] = ICC_SRE_EL2;
```

MSR ICC_SRE_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1001	0b101

```
if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SRE_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        ICC_SRE_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SRE_EL3, Interrupt Controller System Register Enable Register (EL3)

The ICC_SRE_EL3 characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL3.

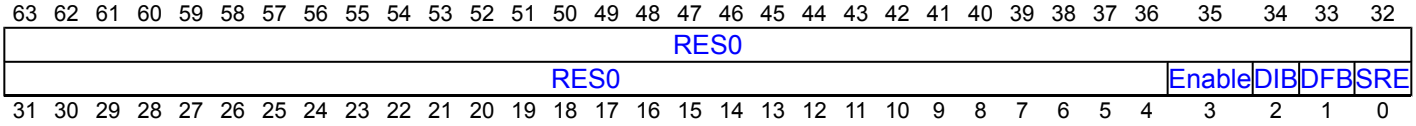
Configuration

This register is present only when GICv3 is implemented, EL3 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICC_SRE_EL3 are UNDEFINED.

Attributes

ICC_SRE_EL3 is a 64-bit register.

Field descriptions



Bits [63:4]

Reserved, RES0.

Enable, bit [3]

Enable. Enables lower Exception level access to ICC_SRE_EL1 and ICC_SRE_EL2.

Enable	Meaning
0b0	EL1 accesses to ICC_SRE_EL1 trap to EL3, unless these accesses are trapped to EL2 as a result of ICC_SRE_EL2.Enable == 0. EL2 accesses to ICC_SRE_EL1 and ICC_SRE_EL2 trap to EL3.
0b1	EL1 accesses to ICC_SRE_EL1 do not trap to EL3. EL2 accesses to ICC_SRE_EL1 and ICC_SRE_EL2 do not trap to EL3.

If ICC_SRE_EL3.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_SRE_EL3.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIB, bit [2]

Disable IRQ bypass.

DIB	Meaning
0b0	IRQ bypass enabled.
0b1	IRQ bypass disabled.

In systems that do not support IRQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

DFB, bit [1]

Disable FIQ bypass.

DFB	Meaning
0b0	FIQ bypass enabled.
0b1	FIQ bypass disabled.

In systems that do not support FIQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

SRE, bit [0]

System Register Enable.

SRE	Meaning
0b0	The memory-mapped interface must be used. Access at EL3 to any ICH_* or ICC_* register other than ICC_SRE_EL1, ICC_SRE_EL2, or ICC_SRE_EL3 is trapped to EL3
0b1	The System register interface to the ICH_* registers and the EL1, EL2, and EL3 ICC_* registers is enabled for EL3.

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If Realm Management Extension is implemented, this field is RAO/WI.

FEAT_GICv3 implementations that do not require GICv2 compatibility might choose to make this bit RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICC_SRE_EL3

This register is always System register accessible.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_SRE_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ICC_SRE_EL3;
```

```
MSR ICC_SRE_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ICC_SRE_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_AP0R<n>_EL2, Interrupt Controller Hyp Active Priorities Group 0 Registers, n = 0 - 3

The ICH_AP0R<n>_EL2 characteristics are:

Purpose

Provides information about Group 0 virtual active priorities for EL2.

Configuration

AArch64 System register ICH_AP0R<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_AP0R<n>[31:0].

This register is present only when GICv3 is implemented, (EL2 is implemented or EL3 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to ICH_AP0R<n>_EL2 are UNDEFINED.

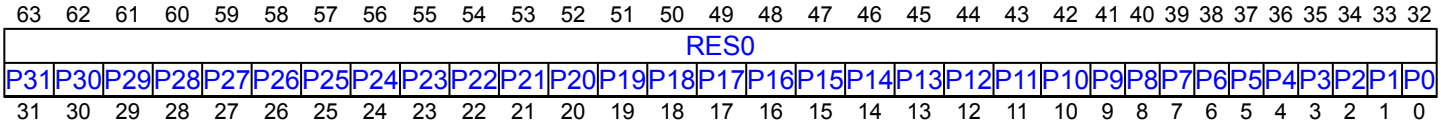
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_AP0R<n>_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

P<x>, bit [x], for x = 31 to 0

Provides the access to the virtual active priorities for Group 0 interrupts. Possible values of each bit are:

P<x>	Meaning
0b0	There is no Group 0 interrupt active with this priority level, or all active Group 0 interrupts with this priority level have undergone priority-drop.
0b1	There is a Group 0 interrupt active with this priority level which has not undergone priority drop.

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP0R0_EL2 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP0R0_EL2 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP0R1_EL2 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP0R0_EL2 in the bits corresponding to 00:Priority[5:1].

- The active state of preemption levels 64 - 126 are held in ICH_AP0R1_EL2 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP0R2_EL2 in the bits corresponding to 10:Priority[5:1].
- The active state of preemption levels 192 - 254 are held in ICH_AP0R3_EL2 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Additional information

Software must ensure that ICH_AP0R<n>_EL2 is 0 for legacy VMs otherwise behavior is UNPREDICTABLE. For more information about support for legacy VMs, see 'Support for legacy operation of VMs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The active priorities for Group 0 and Group 1 interrupts for legacy VMs are held in ICH_AP1R<n>_EL2 and reads and writes to GICV_APR access ICH_AP1R<n>_EL2. This means that ICH_AP0R<n>_EL2 is inaccessible to legacy VMs.

Accessing ICH_AP0R<n>_EL2

ICH_AP0R1_EL2 is implemented only in implementations that support 6 or more bits of preemption. ICH_AP0R2_EL2 and ICH_AP0R3_EL2 are implemented only in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR_EL2.PREbits

Writing to these registers with any value other than the last read value of the register (or `0x00000000` for a newly set up virtual machine) can result in UNPREDICTABLE behavior of the virtual interrupt prioritization system allowing either:

- Virtual interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution at EL1 or EL0.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICH_AP0R<n>_EL2.
- ICH_AP1R<n>_EL2.

Having the bit corresponding to a priority set in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 can result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICH_AP0R<m>_EL2 ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1000	0b0:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x480 + (8 * m)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_AP0R_EL2[m];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_AP0R_EL2[m];
```

MSR ICH_AP0R<m>_EL2, <Xt> ; Where m = 0-3

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1000	0b0:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x480 + (8 * m)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_AP0R_EL2[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_AP0R_EL2[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_AP1R<n>_EL2, Interrupt Controller Hyp Active Priorities Group 1 Registers, n = 0 - 3

The ICH_AP1R<n>_EL2 characteristics are:

Purpose

Provides information about Group 1 virtual active priorities for EL2.

Configuration

AArch64 System register ICH_AP1R<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_AP1R<n>[31:0].

This register is present only when GICv3 is implemented, (EL2 is implemented or EL3 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to ICH_AP1R<n>_EL2 are UNDEFINED.

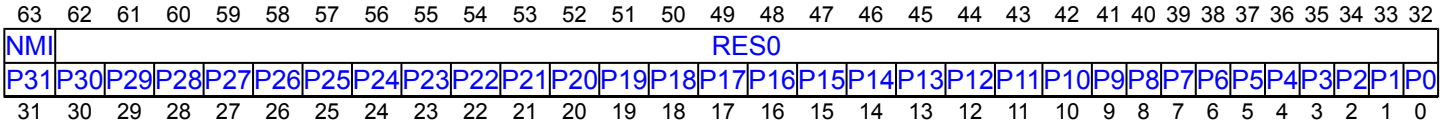
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_AP1R<n>_EL2 is a 64-bit register.

Field descriptions



NMI, bit [63]

When FEAT_GICv3_NMI is implemented and n == 0:

Indicates whether the running virtual priority is from a NMI.

NMI	Meaning
0b0	There is no active Group 1 NMI, or all active Group 1 NMIs have undergone priority drop.
0b1	There is an active Group 1 NMI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bits [62:32]

Reserved, RES0.

P<x>, bit [x], for x = 31 to 0

Group 1 interrupt active priorities. Possible values of each bit are:

P<x>	Meaning
0b0	There is no Group 1 interrupt active with this priority level, or all active Group 1 interrupts with this priority level have undergone priority-drop.
0b1	There is a Group 1 interrupt active with this priority level which has not undergone priority drop.

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP1R0_EL2 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP1R0_EL2 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP1R1_EL2 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP1R0_EL2 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP1R1_EL2 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP1R2_EL2 in the bits corresponding to 10:Priority[5:1].
- The active state of preemption levels 192 - 254 are held in ICH_AP1R3_EL2 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Additional information

This register is always used for legacy VMs, regardless of the group of the virtual interrupt. Reads and writes to GICV_APR<n> access ICH_AP1R<n>_EL2. For more information about support for legacy VMs, see 'Support for legacy operation of VMs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accessing ICH_AP1R<n>_EL2

ICH_AP1R1_EL2 is implemented only in implementations that support 6 or more bits of preemption. ICH_AP1R2_EL2 and ICH_AP1R3_EL2 are implemented only in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR_EL2.PREbits

Writing to these registers with any value other than the last read value of the register (or 0x00000000 for a newly set up virtual machine) can result in UNPREDICTABLE behavior of the virtual interrupt prioritization system allowing either:

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICH_AP1R<m>_EL2 ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x4A0 + (8 * m)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_AP1R_EL2[m];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_AP1R_EL2[m];
```

```
MSR ICH_AP1R<m>_EL2, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x4A0 + (8 * m)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_AP1R_EL2[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_AP1R_EL2[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_EISR_EL2, Interrupt Controller End of Interrupt Status Register

The ICH_EISR_EL2 characteristics are:

Purpose

Indicates which List registers have outstanding EOI maintenance interrupts.

Configuration

AArch64 System register ICH_EISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_EISR[31:0].

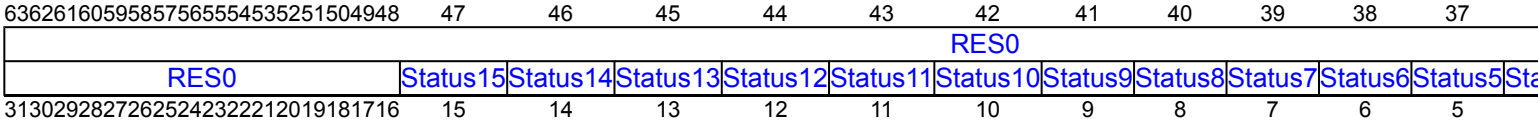
This register is present only when GICv3 is implemented, (EL2 is implemented or EL3 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to ICH_EISR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_EISR_EL2 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

EOI maintenance interrupt status bit for List register <n>:

Status<n>	Meaning
0b0	List register <n>, ICH_LR<n>_EL2, does not have an EOI maintenance interrupt.
0b1	List register <n>, ICH_LR<n>_EL2, has an EOI maintenance interrupt that has not been handled.

For any ICH_LR<n>_EL2, the corresponding status bit is set to 1 if all of the following are true:

- ICH_LR<n>_EL2.State is 0b00.
- ICH_LR<n>_EL2.HW is 0.
- ICH_LR<n>_EL2.EOI (bit [41]) is 1, indicating that when the interrupt corresponding to that List register is deactivated, a maintenance interrupt is asserted.

Otherwise the status bit takes the value 0.

Accessing ICH_EISR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICH_EISR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_EISR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_EISR_EL2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_ELRSR_EL2, Interrupt Controller Empty List Register Status Register

The ICH_ELRSR_EL2 characteristics are:

Purpose

These registers can be used to locate a usable List register when the hypervisor is delivering an interrupt to a VM.

Configuration

AArch64 System register ICH_ELRSR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_ELRSR[31:0].

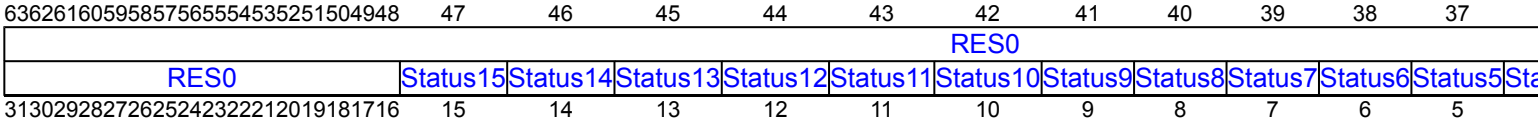
This register is present only when GICv3 is implemented, (EL2 is implemented or EL3 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to ICH_ELRSR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_ELRSR_EL2 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

Status bit for List register <n>, ICH_LR<n>_EL2:

Status<n>	Meaning
0b0	List register ICH_LR<n>_EL2, if implemented, contains a valid interrupt. Using this List register can result in overwriting a valid interrupt.
0b1	List register ICH_LR<n>_EL2 does not contain a valid interrupt. The List register is empty and can be used without overwriting a valid interrupt or losing an EOI maintenance interrupt.

For any List register <n>, the corresponding status bit is set to 1 if ICH_LR<n>_EL2.State is 0b00 and either ICH_LR<n>_EL2.HW is 1 or ICH_LR<n>_EL2.EOI (bit [41]) is 0.

Otherwise the status bit takes the value 0.

Accessing ICH_ELRSR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICH_ELRSR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1011	0b101

```
if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_ELRSR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_ELRSR_EL2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_HCR_EL2, Interrupt Controller Hyp Control Register

The ICH_HCR_EL2 characteristics are:

Purpose

Controls the environment for VMs.

Configuration

AArch64 System register ICH_HCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_HCR[31:0].

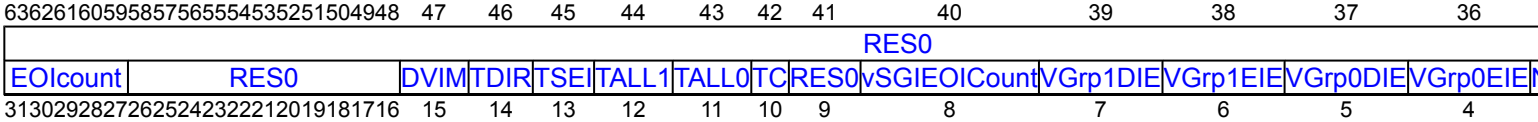
This register is present only when GICv3 is implemented, (EL2 is implemented or EL3 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to ICH_HCR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_HCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

EOIcount, bits [31:27]

This field is incremented whenever a successful write to a virtual EOIR or DIR register would have resulted in a virtual interrupt deactivation. That is either:

- A virtual write to EOIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is zero and no List Register was found.
- A virtual write to DIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is one and no List Register was found.

This allows software to manage more active interrupts than there are implemented List Registers.

It is CONSTRAINED UNPREDICTABLE whether a virtual write to EOIR that does not clear a bit in the Active Priorities registers (ICH_AP0R<n>_EL2/ICH_AP1R<n>_EL2) increments EOIcount. Permitted behaviors are:

- Increment EOIcount.
- Leave EOIcount unchanged.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'00000'`.

Bits [26:16]

Reserved, RES0.

DVIM, bit [15]

When ICH_VTR_EL2.DVIM == 1:

Directly-injected Virtual Interrupt Mask.

DVIM	Meaning
0b0	This control has no effect on the signaling of virtual interrupts.
0b1	Virtual interrupts received via direct-injection are not presented to the virtual CPU interface and not considered when determining the highest priority pending virtual interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

TDIR, bit [14]

When FEAT_GICv3_TDIR is implemented:

Trap EL1 writes to ICC_DIR_EL1 and ICV_DIR_EL1.

TDIR	Meaning
0b0	EL1 writes of ICC_DIR_EL1 and ICV_DIR_EL1 are not trapped to EL2, unless trapped by other mechanisms.
0b1	EL1 writes of ICV_DIR_EL1 are trapped to EL2. It is IMPLEMENTATION DEFINED whether writes of ICC_DIR_EL1 are trapped. Not trapping ICC_DIR_EL1 writes is DEPRECATED.

Arm deprecates not including this trap bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

TSEI, bit [13]

Trap all locally generated SEIs. This bit allows the hypervisor to intercept locally generated SEIs that would otherwise be taken at EL1.

TSEI	Meaning
0b0	Locally generated SEIs do not cause a trap to EL2.
0b1	Locally generated SEIs trap to EL2.

If ICH_VTR_EL2.SEIS is 0, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TALL1, bit [12]

Trap all EL1 accesses to ICC_* and ICV_* System registers for Group 1 interrupts to EL2.

TALL1	Meaning
0b0	EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts proceed as normal.
0b1	EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts trap to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TALL0, bit [11]

Trap all EL1 accesses to ICC_* and ICV_* System registers for Group 0 interrupts to EL2.

TALL0	Meaning
0b0	EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts proceed as normal.
0b1	EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts trap to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TC, bit [10]

Trap all EL1 accesses to System registers that are common to Group 0 and Group 1 to EL2.

TC	Meaning
0b0	EL1 accesses to common registers proceed as normal.
0b1	EL1 accesses to common registers trap to EL2.

This affects accesses to ICC_SGI0R_EL1, ICC_SGI1R_EL1, ICC_ASGI1R_EL1, ICC_CTLR_EL1, ICC_DIR_EL1, ICC_PMR_EL1, ICC_RPR_EL1, ICV_CTLR_EL1, ICV_DIR_EL1, ICV_PMR_EL1, and ICV_RPR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [9]

Reserved, RES0.

vSGIEOICount, bit [8]

When GICv4.1 is implemented:

Controls whether deactivation of virtual SGIs can increment ICH_HCR_EL2.EOIcount

vSGIEOICount	Meaning
0b0	Deactivation of virtual SGIs can increment ICH_HCR_EL2.EOIcount.
0b1	Deactivation of virtual SGIs does not increment ICH_HCR_EL2.EOIcount.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

VGrp1DIE, bit [7]

VM Group 1 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is disabled:

VGrp1DIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp1EIE, bit [6]

VM Group 1 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is enabled:

VGrp1EIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp0DIE, bit [5]

VM Group 0 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is disabled:

VGrp0DIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp0EIE, bit [4]

VM Group 0 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is enabled:

VGrp0EIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

NPIE, bit [3]

No Pending Interrupt Enable. Enables the signaling of a maintenance interrupt when there are no List registers with the State field set to 0b01 (pending):

NPIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled while the List registers contain no interrupts in the pending state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

LRENPIE, bit [2]

List Register Entry Not Present Interrupt Enable. Enables the signaling of a maintenance interrupt while the virtual CPU interface does not have a corresponding valid List register entry for an EOI request:

LRENPIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt is asserted while the EOIcount field is not 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

UIE, bit [1]

Underflow Interrupt Enable. Enables the signaling of a maintenance interrupt when the List registers are empty, or hold only one valid entry:

UIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt is asserted if none, or only one, of the List register entries is marked as a valid interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

En, bit [0]

Enable. Global enable bit for the virtual CPU interface:

En	Meaning
0b0	Virtual CPU interface operation disabled.
0b1	Virtual CPU interface operation enabled.

When this field is set to 0:

- The virtual CPU interface does not signal any maintenance interrupts.
- The virtual CPU interface does not signal any virtual interrupts.
- A read of ICV_IAR0_EL1, ICV_IAR1_EL1, ICV_NMIAR1_EL1, GICV_IAR or GICV_AIAR returns a spurious interrupt ID.

Note

This field is RES0 when SCR_EL3.{NS,EEL2}=={0,0}

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICH_HCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICH_HCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1011	0b000

```
if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x4C0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_HCR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_HCR_EL2;
```

MSR ICH_HCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1011	0b000

```
if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x4C0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_HCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_HCR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_LR<n>_EL2, Interrupt Controller List Registers, n = 0 - 15

The ICH_LR<n>_EL2 characteristics are:

Purpose

Provides interrupt context information for the virtual CPU interface.

Configuration

AArch64 System register ICH_LR<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_LR<n>[31:0].

AArch64 System register ICH_LR<n>_EL2 bits [63:32] are architecturally mapped to AArch32 System register ICH_LRC<n>[31:0].

This register is present only when GICv3 is implemented, (EL2 is implemented or EL3 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to ICH_LR<n>_EL2 are UNDEFINED.

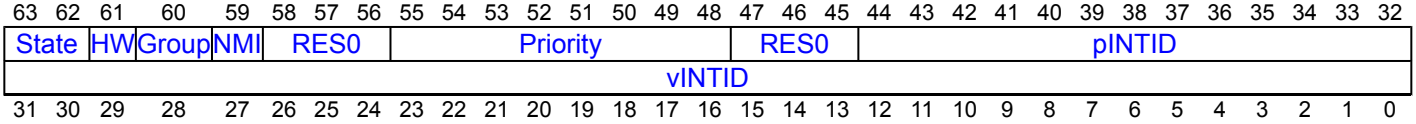
If EL2 is not implemented, this register is RES0 from EL3.

If list register n is not implemented, then accesses to this register are UNDEFINED.

Attributes

ICH_LR<n>_EL2 is a 64-bit register.

Field descriptions



State, bits [63:62]

The state of the interrupt:

State	Meaning
0b00	Invalid (Inactive).
0b01	Pending.
0b10	Active.
0b11	Pending and active.

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. When a virtual interrupt from a List register is acknowledged, the state is updated to the active state regardless of the interrupt type.

Entries in the invalid state are ignored, except for the purpose of generating virtual maintenance interrupts.

For hardware interrupts, the pending and active state is held in the physical Distributor rather than the virtual CPU interface. A hypervisor must only use the pending and active state for software originated interrupts, which are typically associated with virtual devices, or SGIs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HW, bit [61]

Indicates whether this virtual interrupt maps directly to a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt with the ID that the pINTID field indicates.

HW	Meaning
0b0	The interrupt is triggered entirely by software. No notification is sent to the Distributor when the virtual interrupt is deactivated.
0b1	The interrupt maps directly to a hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using the pINTID field from this register to indicate the physical interrupt ID. If ICH_VMCR_EL2.VEOIM is 0, this request corresponds to a write to ICC_EOIR0_EL1 or ICC_EOIR1_EL1. Otherwise, it corresponds to a write to ICC_DIR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Group, bit [60]

Indicates the group for this virtual interrupt.

Group	Meaning
0b0	This is a Group 0 virtual interrupt. ICH_VMCR_EL2.VFIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and ICH_VMCR_EL2.VENG0 enables signaling of this interrupt to the virtual machine.
0b1	This is a Group 1 virtual interrupt, signaled as a virtual IRQ. ICH_VMCR_EL2.VENG1 enables the signaling of this interrupt to the virtual machine. If ICH_VMCR_EL2.VCBPR is 0, then ICC_BPR1_EL1 determines if a pending Group 1 interrupt has sufficient priority to preempt current execution. Otherwise, ICH_LR<n>_EL2 determines preemption.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NMI, bit [59]

When FEAT_GICv3_NMI is implemented:

Indicates whether the virtual priority has the non-maskable property.

NMI	Meaning
0b0	vINTID does not have the non-maskable interrupt property.
0b1	vINTID has the non-maskable interrupt property.

Setting ICH_LR<n>_EL2.NMI to 1 when ICH_LR<n>_EL2.State is not Invalid is CONSTRAINTED UNPREDICTABLE if either ICH_LR<n>_EL2.vINTID indicates an LPI or ICH_LR<n>_EL2.Group is 0.

The permitted behaviors are:

- ICH_LR<n>_EL2.NMI is treated as 0 for all purposes other than a direct read of the register.
- The virtual interrupt is presented with superpriority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [58:56]

Reserved, RES0.

Priority, bits [55:48]

The priority of this interrupt.

It is IMPLEMENTATION DEFINED how many bits of priority are implemented, though at least five bits must be implemented. Unimplemented bits are RES0 and start from bit[48] up to bit[50]. The number of implemented bits can be discovered from ICH_VTR_EL2.PRIbits.

When ICH_LR<n>_EL2.NMI is set to 1, this field is RES0 and the virtual interrupt's priority is treated as `0x00`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:45]

Reserved, RES0.

pINTID, bits [44:32]

Physical INTID, for hardware interrupts.

When ICH_LR<n>_EL2.HW is 0 (there is no corresponding physical interrupt), this field has the following meaning:

- Bits[44:42]: RES0.
- Bit[41]: EOI. If this bit is 1, then when the interrupt identified by vINTID is deactivated, a maintenance interrupt is asserted.
- Bits[40:32]: RES0.

When ICH_LR<n>_EL2.HW is 1 (there is a corresponding physical interrupt):

- This field indicates the physical INTID. This field is only required to implement enough bits to hold a valid value for the implemented INTID size. Any unused higher order bits are RES0.
- When ICC_CTLR_EL1.ExtRange is 0, then bits[44:42] of this field are RES0.
- If the value of pINTID is not a valid INTID, behavior is UNPREDICTABLE. If the value of pINTID indicates a PPI, this field applies to the PPI associated with this same physical PE ID as the virtual CPU interface requesting the deactivation.

A hardware physical identifier is only required in List Registers for interrupts that require deactivation. This means only 13 bits of Physical INTID are required, regardless of the number specified by ICC_CTLR_EL1.IDbits.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

vINTID, bits [31:0]

Virtual INTID of the interrupt.

If the value of vINTID is 1020-1023 and ICH_LR<n>_EL2.State!=`0b00` (Inactive), behavior is UNPREDICTABLE.

Behavior is UNPREDICTABLE if two or more List Registers specify the same vINTID when:

- ICH_LR<n>_EL2.State == `0b01`.
- ICH_LR<n>_EL2.State == `0b10`.
- ICH_LR<n>_EL2.State == `0b11`.

It is IMPLEMENTATION DEFINED how many bits are implemented, though at least 16 bits must be implemented. Unimplemented bits are RES0. The number of implemented bits can be discovered from ICH_VTR_EL2.IDbits.

When ICC_SRE_EL1.SRE == 0, specifying a vINTID in the LPI range is UNPREDICTABLE

Note

When a VM is using memory-mapped access to the GIC, software must ensure that the correct source PE ID is provided in bits[12:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICH_LR<n>_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICH_LR<m>_EL2 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b110:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= NUM_GIC_LIST_REGS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x400 + (8 * m)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_LR_EL2[m];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_LR_EL2[m];
```

```
MSR ICH_LR<m>_EL2, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b110:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= NUM_GIC_LIST_REGS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x400 + (8 * m)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_LR_EL2[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_LR_EL2[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_MISR_EL2, Interrupt Controller Maintenance Interrupt State Register

The ICH_MISR_EL2 characteristics are:

Purpose

Indicates which maintenance interrupts are asserted.

Configuration

AArch64 System register ICH_MISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_MISR[31:0].

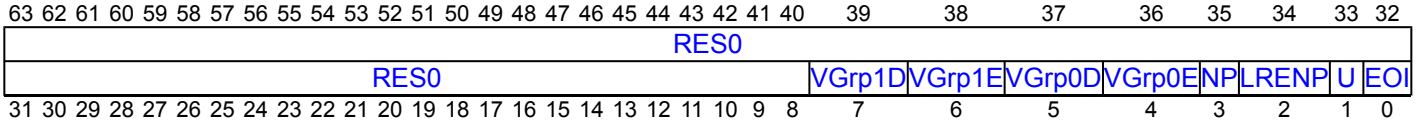
This register is present only when GICv3 is implemented, (EL2 is implemented or EL3 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to ICH_MISR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_MISR_EL2 is a 64-bit register.

Field descriptions



Bits [63:8]

Reserved, RES0.

VGrp1D, bit [7]

vPE Group 1 Disabled.

VGrp1D	Meaning
0b0	vPE Group 1 Disabled maintenance interrupt not asserted.
0b1	vPE Group 1 Disabled maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR_EL2.VGrp1DIE==1 and ICH_VMCR_EL2.VENG1==is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp1E, bit [6]

vPE Group 1 Enabled.

VGrp1E	Meaning
0b0	vPE Group 1 Enabled maintenance interrupt not asserted.
0b1	vPE Group 1 Enabled maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR_EL2.VGrp1EIE==1 and ICH_VMCR_EL2.VENG1==is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp0D, bit [5]

vPE Group 0 Disabled.

VGrp0D	Meaning
0b0	vPE Group 0 Disabled maintenance interrupt not asserted.
0b1	vPE Group 0 Disabled maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR_EL2.VGrp0DIE==1 and ICH_VMCR_EL2.VENG0==0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp0E, bit [4]

vPE Group 0 Enabled.

VGrp0E	Meaning
0b0	vPE Group 0 Enabled maintenance interrupt not asserted.
0b1	vPE Group 0 Enabled maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR_EL2.VGrp0EIE==1 and ICH_VMCR_EL2.VENG0==1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

NP, bit [3]

No Pending.

NP	Meaning
0b0	No Pending maintenance interrupt not asserted.
0b1	No Pending maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR_EL2.NPIE==1 and no List register is in pending state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

LRENP, bit [2]

List Register Entry Not Present.

LRENP	Meaning
0b0	List Register Entry Not Present maintenance interrupt not asserted.
0b1	List Register Entry Not Present maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR_EL2.LRENPIE==1 and ICH_HCR_EL2.EOIcount is nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

U, bit [1]

Underflow.

U	Meaning
0b0	Underflow maintenance interrupt not asserted.
0b1	Underflow maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR_EL2.UIE==1 and zero or one of the List register entries are marked as a valid interrupt, that is, if the corresponding ICH_LR<n>_EL2.State bits do not equal 0x0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

EOI, bit [0]

End Of Interrupt.

EOI	Meaning
0b0	End Of Interrupt maintenance interrupt not asserted.
0b1	End Of Interrupt maintenance interrupt asserted.

This maintenance interrupt is asserted when at least one bit in ICH_EISR_EL2 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Additional information

The U and NP bits do not include the status of any pending/active 'VSet (IRI)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069) packets because these bits control generation of interrupts that allow software management of the contents of the List Registers (which are not affected by 'VSet (IRI)' packets).

Accessing ICH_MISR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICH_MISR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1011	0b010

```
if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_MISR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_MISR_EL2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register

The ICH_VMCR_EL2 characteristics are:

Purpose

Enables the hypervisor to save and restore the virtual machine view of the GIC state.

Configuration

AArch64 System register ICH_VMCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_VMCR[31:0].

This register is present only when GICv3 is implemented, (EL2 is implemented or EL3 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to ICH_VMCR_EL2 are UNDEFINED.

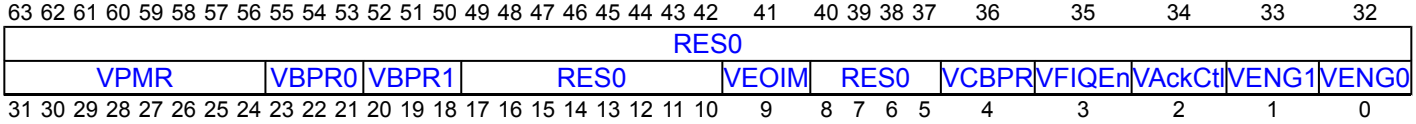
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_VMCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

VPMR, bits [31:24]

Virtual Priority Mask. The priority mask level for the virtual CPU interface. If the priority of a pending virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

This field is an alias of ICV_PMR_EL1.Priority.

VBPR0, bits [23:21]

Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if ICH_VMCR_EL2.VCBPR == 1.

This field is an alias of ICV_BPR0_EL1.BinaryPoint.

The minimum value of this field is determined by ICH_VTR_EL2.PREbits. An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

VBPR1, bits [20:18]

Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption if ICH_VMCR_EL2.VCBPR == 0.

This field is an alias of ICV_BPR1_EL1.BinaryPoint.

This field is always accessible to EL2 accesses, regardless of the setting of the ICH_VMCR_EL2.VCBPR field.

For Non-secure writes, the minimum value of this field is the minimum value of ICH_VMCR_EL2.VBPR0 plus one.

For Secure writes, the minimum value of this field is the minimum value of ICH_VMCR_EL2.VBPR0.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

Bits [17:10]

Reserved, RES0.

VEOIM, bit [9]

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

VEOIM	Meaning
0b0	ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR_EL1 are UNPREDICTABLE.
0b1	ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide priority drop functionality only. ICV_DIR_EL1 provides interrupt deactivation functionality.

This bit is an alias of ICV_CTLR_EL1.EOImode.

Bits [8:5]

Reserved, RES0.

VCBPR, bit [4]

Virtual Common Binary Point Register. Possible values of this bit are:

VCBPR	Meaning
0b0	ICV_BPR1_EL1 determines the preemption group for virtual Group 1 interrupts.
0b1	Reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 plus one, saturated to 0b111. Writes to ICV_BPR1_EL1 are ignored.

This field is an alias of ICV_CTLR_EL1.CBPR.

VFIQEn, bit [3]

Virtual FIQ enable. Possible values of this bit are:

VFIQEn	Meaning
0b0	Group 0 virtual interrupts are presented as virtual IRQs.
0b1	Group 0 virtual interrupts are presented as virtual FIQs.

This bit is an alias of GICV_CTLR.FIQEn.

In implementations where the Non-secure copy of ICC_SRE_EL1.SRE is always 1, this bit is RES1.

VAckCtl, bit [2]

Virtual AckCtl. Possible values of this bit are:

VAckCtl	Meaning
0b0	If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an INTID of 1022.
0b1	If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the INTID of the corresponding interrupt.

This bit is an alias of GICV_CTLR.AckCtl.

This field is supported for backwards compatibility with GICv2. Arm deprecates the use of this field.

In implementations where the Non-secure copy of ICC_SRE_EL1.SRE is always 1, this bit is RES0.

VENG1, bit [1]

Virtual Group 1 interrupt enable. Possible values of this bit are:

VENG1	Meaning
0b0	Virtual Group 1 interrupts are disabled.
0b1	Virtual Group 1 interrupts are enabled.

This bit is an alias of ICV_IGRPEN1_EL1.Enable.

VENG0, bit [0]

Virtual Group 0 interrupt enable. Possible values of this bit are:

VENG0	Meaning
0b0	Virtual Group 0 interrupts are disabled.
0b1	Virtual Group 0 interrupts are enabled.

This bit is an alias of ICV_IGRPEN0_EL1.Enable.

Accessing ICH_VMCR_EL2

When EL2 is using System register access, EL1 using either System register or memory-mapped access must be supported.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICH_VMCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1011	0b111

```
if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x4C8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_VMCR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_VMCR_EL2;
```

MSR ICH_VMCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1011	0b111

```
if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x4C8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_VMCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_VMCR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_VTR_EL2, Interrupt Controller VGIC Type Register

The ICH_VTR_EL2 characteristics are:

Purpose

Reports supported GIC virtualization features.

Configuration

AArch64 System register ICH_VTR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_VTR[31:0].

This register is present only when GICv3 is implemented, (EL2 is implemented or EL3 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to ICH_VTR_EL2 are UNDEFINED.

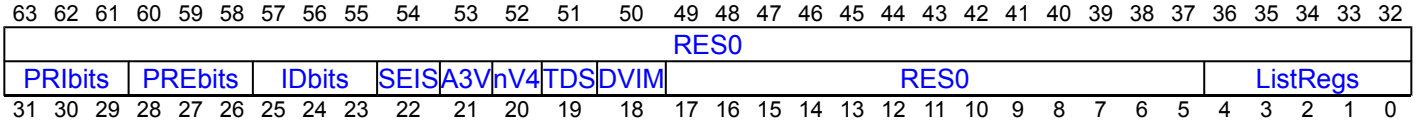
If EL2 is not implemented, all bits in this register are RES0 from EL3, except for nV4, which is RES1 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_VTR_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

PRIbits, bits [31:29]

Priority bits. Indicates the number of virtual priority bits implemented, minus one.

An implementation must implement at least 32 levels of virtual priority (5 priority bits).

This field is an alias of ICV_CTLR_EL1.PRIbits.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PRIbits	Meaning
0b100..0b110	The number of virtual priority bits implemented, minus one.

Access to this field is **RO**.

PREbits, bits [28:26]

Preemption bits. Indicates the number of virtual preemption bits implemented, minus one.

An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).

The value of this field must be less than or equal to the value of ICH_VTR_EL2.PRIbits.

This field determines the minimum value of ICH_VMCR_EL2.VBPR0.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PREbits	Meaning
0b000..0b110	The number of virtual preemption bits implemented, minus one.

Access to this field is **RO**.

IDbits, bits [25:23]

The number of virtual interrupt identifier bits supported:

The value of this field is an IMPLEMENTATION DEFINED choice of:

IDbits	Meaning
0b000	16 bits.
0b001	24 bits.

All other values are reserved.

This field is an alias of ICV_CTLR_EL1.IDbits.

Access to this field is **RO**.

SEIS, bit [22]

SEI Support. Indicates whether the virtual CPU interface supports generation of SEIs:

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEIS	Meaning
0b0	The virtual CPU interface logic does not support generation of SEIs.
0b1	The virtual CPU interface logic supports generation of SEIs.

This bit is an alias of ICV_CTLR_EL1.SEIS.

Access to this field is **RO**.

A3V, bit [21]

Affinity 3 Valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

A3V	Meaning
0b0	The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.
0b1	The virtual CPU interface logic supports nonzero values of Affinity 3 in SGI generation System registers.

This bit is an alias of ICV_CTLR_EL1.A3V.

Access to this field is **RO**.

nV4, bit [20]

Direct injection of virtual interrupts not supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

nV4	Meaning
0b0	The CPU interface logic supports direct injection of virtual interrupts.
0b1	The CPU interface logic does not support direct injection of virtual interrupts.

In GICv3, the only permitted value is 0b1.

Access to this field is **RO**.

TDS, bit [19]

Separate trapping of EL1 writes to ICV_DIR_EL1 supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TDS	Meaning
0b0	Implementation does not support ICH_HCR_EL2.TDIR.
0b1	Implementation supports ICH_HCR_EL2.TDIR.

FEAT_GICv3_TDIR implements the functionality added by the value 0b1.

Access to this field is **RO**.

DVIM, bit [18]

Masking of directly-injected virtual interrupts.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DVIM	Meaning
0b0	Masking of Directly-injected Virtual Interrupts not supported.
0b1	Masking of Directly-injected Virtual Interrupts is supported.

When a PE implements the Realm Management Extension, this field is RAO/WI.

Access to this field is **RO**.

Bits [17:5]

Reserved, RES0.

ListRegs, bits [4:0]

List Registers. Indicates the number of List registers implemented, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ListRegs	Meaning
0b00000..0b01111	The number of List registers implemented, minus one.

Access to this field is **RO**.

Accessing ICH_VTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICH_VTR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b1011	0b001

```
if !(IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) || HaveEL(EL3)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ICH_VTR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICH_VTR_EL2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_AP0R<n>_EL1, Interrupt Controller Virtual Active Priorities Group 0 Registers, n = 0 - 3

The ICV_AP0R<n>_EL1 characteristics are:

Purpose

Provides information about virtual Group 0 active priorities.

Configuration

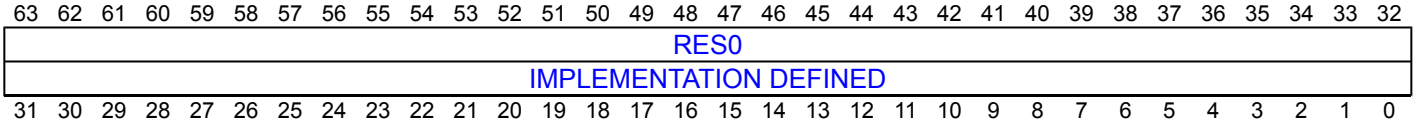
AArch64 System register ICV_AP0R<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_AP0R<n>[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_AP0R<n>_EL1 are UNDEFINED.

Attributes

ICV_AP0R<n>_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value `0x00000000` is consistent with no interrupts being active.

Accessing ICV_AP0R<n>_EL1

Writing to these registers with any value other than the last read value of the register (or `0x00000000` when there are no Group 0 active priorities) might result in UNPREDICTABLE behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP0R1_EL1 is implemented only in implementations that support 6 or more bits of priority. ICV_AP0R2_EL1 and ICV_AP0R3_EL1 are implemented only in implementations that support 7 bits of priority. Unimplemented registers are UNDEFINED.

Writing to the active priority registers in any order other than the following order might result in UNPREDICTABLE behavior of the interrupt prioritization system:

- ICV_AP0R<n>_EL1.
- ICV_AP1R<n>_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_AP0R<m>_EL1 ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b1:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_AP0R_EL1[m];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_AP0R_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_AP0R_EL1[m];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_AP0R_EL1[m];
```

```
MSR ICC_AP0R<m>_EL1, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b1:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_AP0R_EL1[m] = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_AP1R<n>_EL1, Interrupt Controller Virtual Active Priorities Group 1 Registers, n = 0 - 3

The ICV_AP1R<n>_EL1 characteristics are:

Purpose

Provides information about virtual Group 1 active priorities.

Configuration

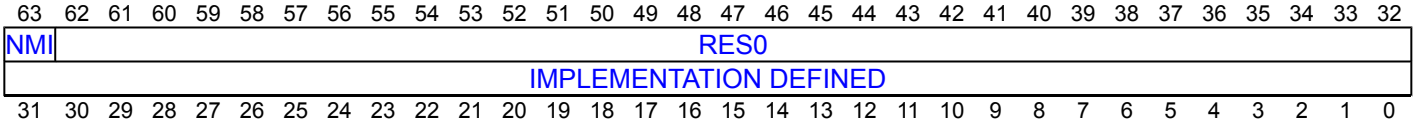
AArch64 System register ICV_AP1R<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_AP1R<n>[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_AP1R<n>_EL1 are UNDEFINED.

Attributes

ICV_AP1R<n>_EL1 is a 64-bit register.

Field descriptions



NMI, bit [63]

When FEAT_GICv3_NMI is implemented and n == 0:

Indicates whether the running priority is from a NMI.

NMI	Meaning
0b0	There is no active Group 1 NMI, or all active Group 1 NMIs have undergone priority-drop.
0b1	There is an active Group 1 NMI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bits [62:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value `0x00000000` is consistent with no interrupts being active.

Accessing ICV_AP1R<n>_EL1

Writing to these registers with any value other than the last read value of the register (or `0x00000000` when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP1R1_EL1 is implemented only in implementations that support 6 or more bits of priority. ICV_AP1R2_EL1 and ICV_AP1R3_EL1 are implemented only in implementations that support 7 bits of priority. Unimplemented registers are UNDEFINED.

Writing to the active priority registers in any order other than the following order might result in UNPREDICTABLE behavior of the interrupt prioritization system:

- ICV_AP0R<n>_EL1.
- ICV_AP1R<n>_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_AP1R<m>_EL1 ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_AP1R_EL1[m];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_AP1R_EL1_S[m];
        else
            X[t, 64] = ICC_AP1R_EL1_NS[m];
    else
        X[t, 64] = ICC_AP1R_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_AP1R_EL1_S[m];
        else
            X[t, 64] = ICC_AP1R_EL1_NS[m];
    else
        X[t, 64] = ICC_AP1R_EL1[m];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_AP1R_EL1_S[m];
        else
            X[t, 64] = ICC_AP1R_EL1_NS[m];
```

```
MSR ICC_AP1R<m>_EL1, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(op2<1:0>);

if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_AP1R_EL1[m] = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[m] = X[t, 64];
        else
            ICC_AP1R_EL1_NS[m] = X[t, 64];
    else
        ICC_AP1R_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[m] = X[t, 64];
        else
            ICC_AP1R_EL1_NS[m] = X[t, 64];
    else
        ICC_AP1R_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[m] = X[t, 64];
        else
            ICC_AP1R_EL1_NS[m] = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_BPR0_EL1, Interrupt Controller Virtual Binary Point Register 0

The ICV_BPR0_EL1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 0 interrupt preemption.

Configuration

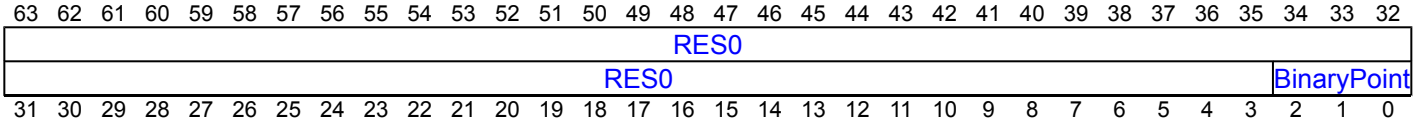
AArch64 System register ICV_BPR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_BPR0[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_BPR0_EL1 are UNDEFINED.

Attributes

ICV_BPR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

Binary point value	Group priority field	Subpriority field	Field with binary point
0	[7:1]	[0]	ggggggg.s
1	[7:2]	[1:0]	gggggg.ss
2	[7:3]	[2:0]	ggggg.sss
3	[7:4]	[3:0]	gggg.ssss
4	[7:5]	[4:0]	ggg.sssss
5	[7:6]	[5:0]	gg.ssssss
6	[7]	[6:0]	g.sssssss
7	No preemption	[7:0]	.sssssssss

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_BPR0_EL1

The minimum binary point value is derived from the number of implemented preemption bits, as shown in the following table:

Number of implemented preemption bits	Minimum value of BPR0
7	0
6	1
5	2

The number of implemented preemption bits is indicated by ICH_VTR_EL2.PREbits.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_BPR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_BPR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_BPR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_BPR0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_BPR0_EL1;
```

```
MSR ICC_BPR0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_BPR0_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_BPR1_EL1, Interrupt Controller Virtual Binary Point Register 1

The ICV_BPR1_EL1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 1 interrupt preemption.

Configuration

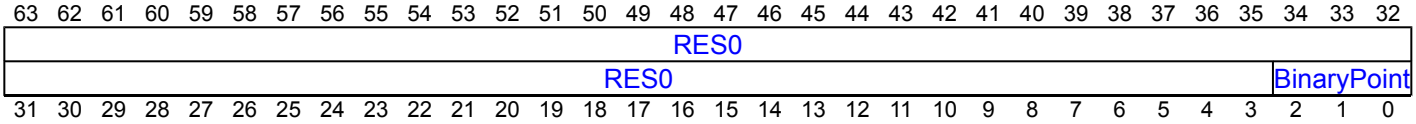
AArch64 System register ICV_BPR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_BPR1[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_BPR1_EL1 are UNDEFINED.

Attributes

ICV_BPR1_EL1 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

If the GIC is configured to use separate binary point fields for Group 0 and Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

For more information about priorities, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

An attempt to program this field to a value less than the minimum value sets the field to the minimum value.

If ICV_CTLR_EL1.CBPR is set to 1, Non-secure EL1 reads return ICV_BPR0_EL1 + 1 saturated to `0b111`. Non-secure EL1 writes are ignored.

If ICV_CTLR_EL1.CBPR is set to 1, Secure EL1 reads return ICV_BPR0_EL1. Secure EL1 writes modify ICV_BPR0_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_BPR1_EL1

For Non-secure writes, the minimum value of this field is the minimum value of ICH_VMCR_EL2.VBPR0 plus one.

For Secure writes, the minimum value of this field is the minimum value of ICH_VMCR_EL2.VBPR0.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_BPR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_BPR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_BPR1_EL1_S;
        else
            X[t, 64] = ICC_BPR1_EL1_NS;
    else
        X[t, 64] = ICC_BPR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_BPR1_EL1_S;
        else
            X[t, 64] = ICC_BPR1_EL1_NS;
    else
        X[t, 64] = ICC_BPR1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_BPR1_EL1_S;
        else
            X[t, 64] = ICC_BPR1_EL1_NS;
```

```
MSR ICC_BPR1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_BPR1_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t, 64];
        else
            ICC_BPR1_EL1_NS = X[t, 64];
    else
        ICC_BPR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t, 64];
        else
            ICC_BPR1_EL1_NS = X[t, 64];
    else
        ICC_BPR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t, 64];
        else
            ICC_BPR1_EL1_NS = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_CTLR_EL1, Interrupt Controller Virtual Control Register

The ICV_CTLR_EL1 characteristics are:

Purpose

Controls aspects of the behavior of the GIC virtual CPU interface and provides information about the features implemented.

Configuration

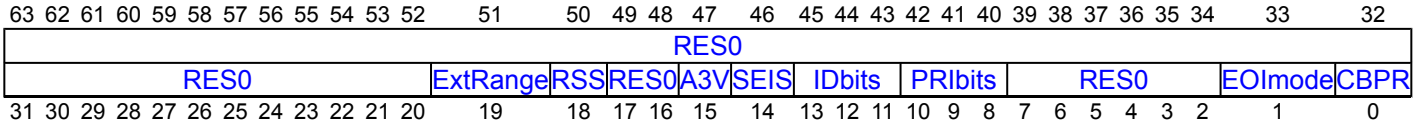
AArch64 System register ICV_CTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_CTLR[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_CTLR_EL1 are UNDEFINED.

Attributes

ICV_CTLR_EL1 is a 64-bit register.

Field descriptions



Bits [63:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

The value of this field is an IMPLEMENTATION DEFINED choice of:

ExtRange	Meaning
0b0	CPU interface does not support INTIDs in the range 1024..8191. • Behavior is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface. **Note** Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.
0b1	CPU interface supports INTIDs in the range 1024..8191 • All INTIDs in the range 1024..8191 are treated as requiring deactivation.

ICV_CTLR_EL1.ExtRange is an alias of ICC_CTLR_EL1.ExtRange.

Access to this field is **RO**.

RSS, bit [18]

Range Selector Support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RSS	Meaning
0b0	Targeted SGIs with affinity level 0 values of 0 - 15 are supported.
0b1	Targeted SGIs with affinity level 0 values of 0 - 255 are supported.

This bit is read-only.

Access to this field is **RO**.

Bits [17:16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored.

The value of this field is an IMPLEMENTATION DEFINED choice of:

A3V	Meaning
0b0	The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.
0b1	The virtual CPU interface logic supports nonzero values of Affinity 3 in SGI generation System registers.

Access to this field is **RO**.

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the virtual CPU interface supports local generation of SEIs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEIS	Meaning
0b0	The virtual CPU interface logic does not support local generation of SEIs.
0b1	The virtual CPU interface logic supports local generation of SEIs.

Access to this field is **RO**.

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. Indicates the number of virtual interrupt identifier bits supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IDbits	Meaning
0b000	16 bits.
0b001	24 bits.

All other values are reserved.

Access to this field is **RO**.

PRIbits, bits [10:8]

Indicates the number of virtual priority bits implemented.

An implementation must implement at least 32 levels of virtual priority (5 priority bits).

The division between group priority and subpriority is defined in the binary point registers ICV_BPR0_EL1 and ICV_BPR1_EL1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PRIbits	Meaning
0b100..0b110	The number of virtual priority bits implemented, minus one.

Access to this field is **RO**.

Bits [7:2]

Reserved, RES0.

EOImode, bit [1]

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

EOImode	Meaning
0b0	ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR_EL1 are UNPREDICTABLE.
0b1	ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide priority drop functionality only. ICV_DIR_EL1 provides interrupt deactivation functionality.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR, bit [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both virtual Group 0 and virtual Group 1 interrupts:

CBPR	Meaning
0b0	ICV_BPR1_EL1 determines the preemption group for virtual Group 1 interrupts.
0b1	Non-secure reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 plus one, saturated to 0b111. Non-secure writes to ICV_BPR1_EL1 are ignored. Secure reads of ICV_BPR1_EL1 return ICV_BPR0_EL1. Secure writes of ICV_BPR1_EL1 modify ICV_BPR0_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_CTLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_CTLR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_CTLR_EL1;
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_CTLR_EL1;
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_CTLR_EL1_S;
        else
            X[t, 64] = ICC_CTLR_EL1_NS;
    else
        X[t, 64] = ICC_CTLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_CTLR_EL1_S;
        else
            X[t, 64] = ICC_CTLR_EL1_NS;
    else
        X[t, 64] = ICC_CTLR_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_CTLR_EL1_S;
        else
            X[t, 64] = ICC_CTLR_EL1_NS;
```

```
MSR ICC_CTLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_CTLR_EL1 = X[t, 64];
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_CTLR_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t, 64];
        else
            ICC_CTLR_EL1_NS = X[t, 64];
    else
        ICC_CTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t, 64];
        else
            ICC_CTLR_EL1_NS = X[t, 64];
    else
        ICC_CTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t, 64];
        else
            ICC_CTLR_EL1_NS = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_DIR_EL1, Interrupt Controller Deactivate Virtual Interrupt Register

The ICV_DIR_EL1 characteristics are:

Purpose

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified virtual interrupt.

Configuration

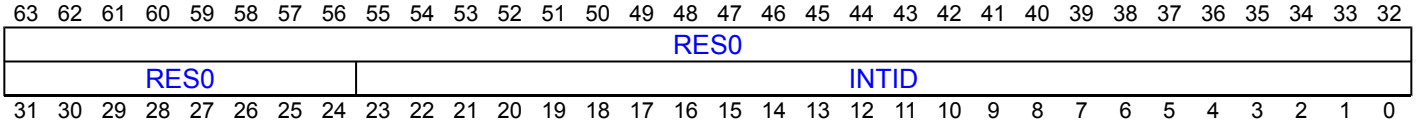
AArch64 System register ICV_DIR_EL1 bits [31:0] performs the same function as AArch32 System register ICV_DIR[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_DIR_EL1 are UNDEFINED.

Attributes

ICV_DIR_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the virtual interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_DIR_EL1

When EOImode == 0, writes are ignored. In systems supporting system error generation, an implementation might generate an SEI.

Accesses to this register use the following encodings in the System register encoding space:

```
MSR ICC_DIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1011	0b001

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TDIR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_DIR_EL1 = X[t, 64];
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_DIR_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_EOIR0_EL1, Interrupt Controller Virtual End Of Interrupt Register 0

The ICV_EOIR0_EL1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 0 interrupt.

Configuration

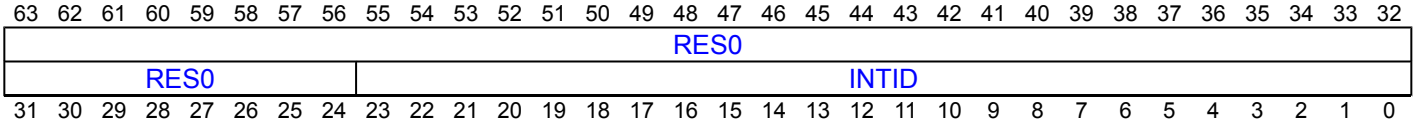
AArch64 System register ICV_EOIR0_EL1 bits [31:0] performs the same function as AArch32 System register ICV_EOIR0[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_EOIR0_EL1 are UNDEFINED.

Attributes

ICV_EOIR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding ICV_IAR0_EL1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the ICV_CTLR_EL1.EOImode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the ICV_CTLR_EL1.EOImode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to ICV_DIR_EL1 to deactivate the virtual interrupt.

Accessing ICV_EOIR0_EL1

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICV_IAR0_EL1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings in the System register encoding space:

```
MSR ICC_EOIR0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2

0b11	0b000	0b1100	0b1000	0b001

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_EOIR0_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_EOIR1_EL1, Interrupt Controller Virtual End Of Interrupt Register 1

The ICV_EOIR1_EL1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 1 interrupt.

Configuration

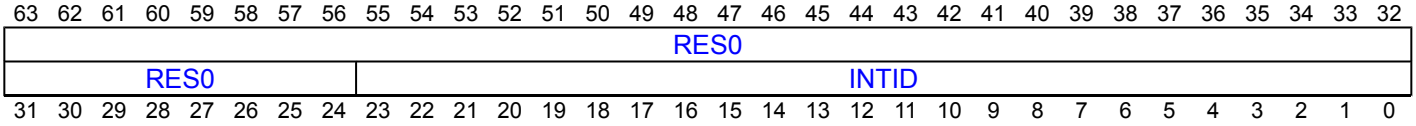
AArch64 System register ICV_EOIR1_EL1 bits [31:0] performs the same function as AArch32 System register ICV_EOIR1[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_EOIR1_EL1 are UNDEFINED.

Attributes

ICV_EOIR1_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding ICV_IAR1_EL1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the ICV_CTLR_EL1.EOImode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the ICV_CTLR_EL1.EOImode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to ICV_DIR_EL1 to deactivate the virtual interrupt.

Accessing ICV_EOIR1_EL1

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICV_IAR1_EL1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings in the System register encoding space:

```
MSR ICC_EOIR1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2

0b11	0b000	0b1100	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_EOIR1_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_HPPIR0_EL1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

The ICV_HPPIR0_EL1 characteristics are:

Purpose

Indicates the highest priority pending virtual Group 0 interrupt on the virtual CPU interface.

Configuration

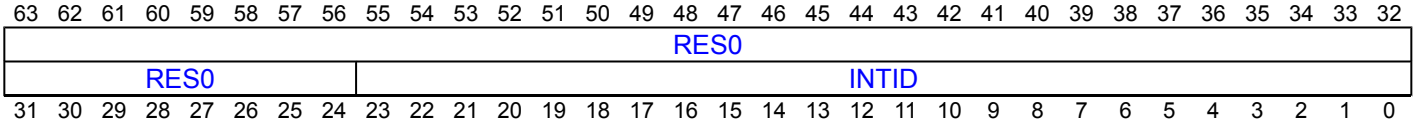
AArch64 System register ICV_HPPIR0_EL1 bits [31:0] performs the same function as AArch32 System register ICV_HPPIR0[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_HPPIR0_EL1 are UNDEFINED.

Attributes

ICV_HPPIR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_HPPIR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_HPPIR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b010

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_HPPIR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_HPPIR1_EL1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

The ICV_HPPIR1_EL1 characteristics are:

Purpose

Indicates the highest priority pending virtual Group 1 interrupt on the virtual CPU interface.

Configuration

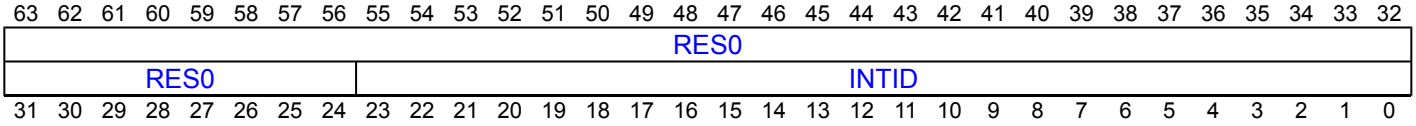
AArch64 System register ICV_HPPIR1_EL1 bits [31:0] performs the same function as AArch32 System register ICV_HPPIR1[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_HPPIR1_EL1 are UNDEFINED.

Attributes

ICV_HPPIR1_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_HPPIR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_HPPIR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_HPPIR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_HPPIR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IAR0_EL1, Interrupt Controller Virtual Interrupt Acknowledge Register 0

The ICV_IAR0_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 0 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch64 System register ICV_IAR0_EL1 bits [31:0] performs the same function as AArch32 System register ICV_IAR0[31:0].

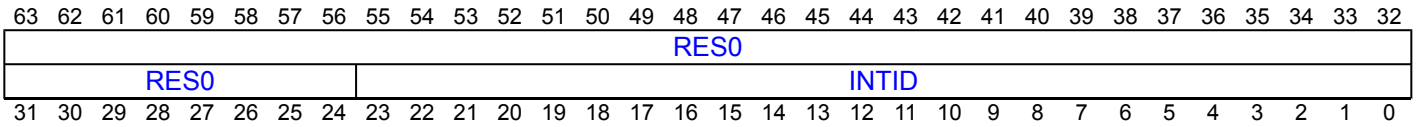
This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_IAR0_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronizing when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_IAR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_IAR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_IAR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_IAR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IAR1_EL1, Interrupt Controller Virtual Interrupt Acknowledge Register 1

The ICV_IAR1_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch64 System register ICV_IAR1_EL1 bits [31:0] performs the same function as AArch32 System register ICV_IAR1[31:0].

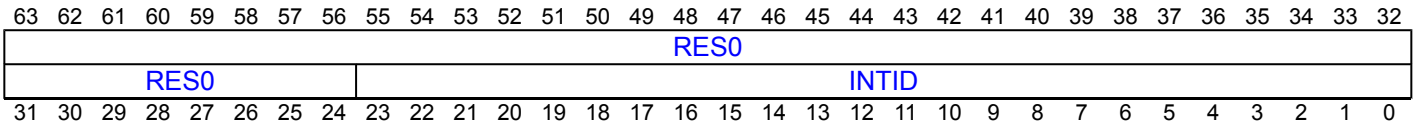
This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_IAR1_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronizing when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_IAR1_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_IAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_IAR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_IAR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IAR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IGRPEN0_EL1, Interrupt Controller Virtual Interrupt Group 0 Enable Register

The ICV_IGRPEN0_EL1 characteristics are:

Purpose

Controls whether virtual Group 0 interrupts are enabled or not.

Configuration

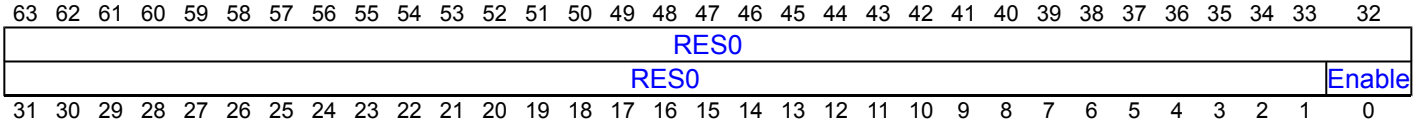
AArch64 System register ICV_IGRPEN0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_IGRPEN0[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_IGRPEN0_EL1 are UNDEFINED.

Attributes

ICV_IGRPEN0_EL1 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

Enable, bit [0]

Enables virtual Group 0 interrupts.

Enable	Meaning
0b0	Virtual Group 0 interrupts are disabled.
0b1	Virtual Group 0 interrupts are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_IGRPEN0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_IGRPEN0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_IGRPEN0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IGRPEN0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IGRPEN0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_IGRPEN0_EL1;
```

```
MSR ICC_IGRPEN0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_IGRPEN0_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IGRPEN1_EL1, Interrupt Controller Virtual Interrupt Group 1 Enable Register

The ICV_IGRPEN1_EL1 characteristics are:

Purpose

Controls whether virtual Group 1 interrupts are enabled for the current Security state.

Configuration

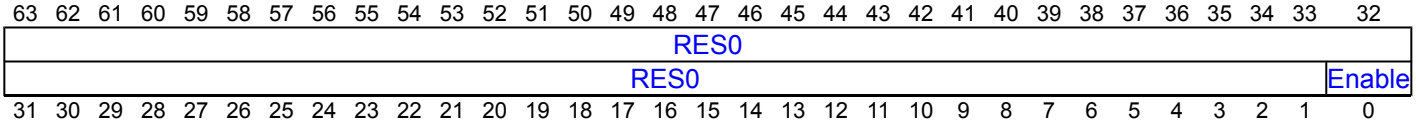
AArch64 System register ICV_IGRPEN1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_IGRPEN1[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_IGRPEN1_EL1 are UNDEFINED.

Attributes

ICV_IGRPEN1_EL1 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

Enable, bit [0]

Enables virtual Group 1 interrupts.

Enable	Meaning
0b0	Virtual Group 1 interrupts are disabled.
0b1	Virtual Group 1 interrupts are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_IGRPEN1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_IGRPEN1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_IGRPEN1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_IGRPEN1_EL1_S;
        else
            X[t, 64] = ICC_IGRPEN1_EL1_NS;
    else
        X[t, 64] = ICC_IGRPEN1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_IGRPEN1_EL1_S;
        else
            X[t, 64] = ICC_IGRPEN1_EL1_NS;
    else
        X[t, 64] = ICC_IGRPEN1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            X[t, 64] = ICC_IGRPEN1_EL1_S;
        else
            X[t, 64] = ICC_IGRPEN1_EL1_NS;
```

```
MSR ICC_IGRPEN1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_IGRPEN1_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t, 64];
        else
            ICC_IGRPEN1_EL1_NS = X[t, 64];
    else
        ICC_IGRPEN1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t, 64];
        else
            ICC_IGRPEN1_EL1_NS = X[t, 64];
    else
        ICC_IGRPEN1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t, 64];
        else
            ICC_IGRPEN1_EL1_NS = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_NMIAR1_EL1, Interrupt Controller Virtual Non-maskable Interrupt Acknowledge Register 1

The ICV_NMIAR1_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

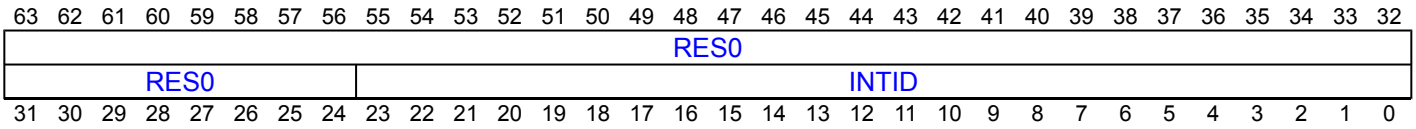
This register is present only when FEAT_GICv3_NMI is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_NMIAR1_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronizing when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_NMIAR1_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that virtual interrupt has the Non-maskable property and is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_NMIAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_NMIAR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1001	0b101

```
if !(IsFeatureImplemented(FEAT_GICv3_NMI) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if SCTLR_EL1.NMI == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_NMIAR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_NMIAR1_EL1;
elsif PSTATE.EL == EL2 then
    if SCTLR_EL2.NMI == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_NMIAR1_EL1;
elsif PSTATE.EL == EL3 then
    if SCTLR_EL3.NMI == '0' then
        UNDEFINED;
    elsif ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_NMIAR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_PMR_EL1, Interrupt Controller Virtual Interrupt Priority Mask Register

The ICV_PMR_EL1 characteristics are:

Purpose

Provides a virtual interrupt priority filter. Only virtual interrupts with a higher priority than the value in this register are signaled to the PE.

Configuration

AArch64 System register ICV_PMR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_PMR[31:0].

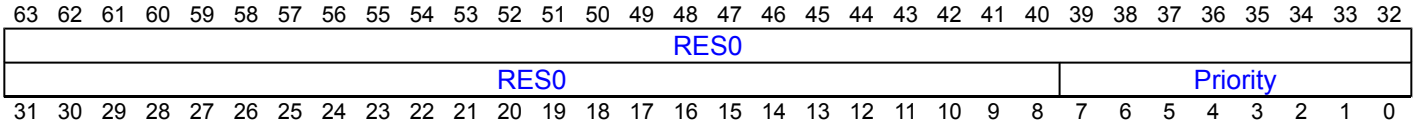
This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_PMR_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronizing. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_PMR_EL1 is a 64-bit register.

Field descriptions



Bits [63:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the virtual CPU interface. If the priority of a virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

The possible priority field values are as follows:

Implemented priority bits	Possible priority field values	Number of priority levels
[7:0]	0x00-0xFF (0-255), all values	256
[7:1]	0x00-0xFE (0-254), even values only	128
[7:2]	0x00-0xFC (0-252), in steps of 4	64
[7:3]	0x00-0xF8 (0-248), in steps of 8	32
[7:4]	0x00-0xF0 (0-240), in steps of 16	16

Unimplemented priority bits are RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_PMR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_PMR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_PMR_EL1;
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_PMR_EL1;
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_PMR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_PMR_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_PMR_EL1;
```

```
MSR ICC_PMR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_PMR_EL1 = X[t, 64];
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_PMR_EL1 = X[t, 64];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_RPR_EL1, Interrupt Controller Virtual Running Priority Register

The ICV_RPR_EL1 characteristics are:

Purpose

Indicates the Running priority of the virtual CPU interface.

Configuration

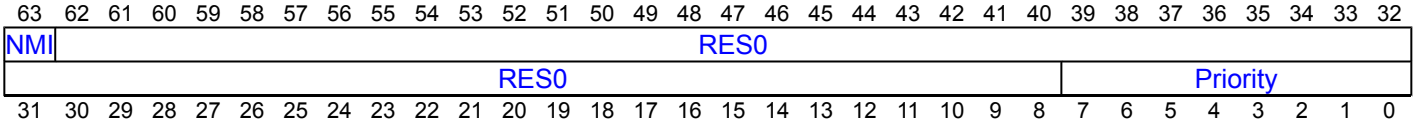
AArch64 System register ICV_RPR_EL1 bits [31:0] performs the same function as AArch32 System register ICV_RPR[31:0].

This register is present only when GICv3 is implemented, EL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to ICV_RPR_EL1 are UNDEFINED.

Attributes

ICV_RPR_EL1 is a 64-bit register.

Field descriptions



NMI, bit [63]

When FEAT_GICv3_NMI is implemented:

Indicates whether the running priority is from a NMI.

NMI	Meaning
0b0	There is no active Group 1 NMI, or all active Group 1 NMIs have undergone priority drop.
0b1	There is an active Group 1 NMI.

Otherwise:

Reserved, RES0.

Bits [62:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the virtual CPU interface. This is the group priority of the current active virtual interrupt.

If there are no active interrupts on the virtual CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

Note

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

Accessing ICV_RPR_EL1

If there are no active interrupts on the virtual CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ICC_RPR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_GICv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        X[t, 64] = ICV_RPR_EL1;
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        X[t, 64] = ICV_RPR_EL1;
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_RPR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_RPR_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ICC_RPR_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64AFR0_EL1, AArch64 Auxiliary Feature Register 0

The ID_AA64AFR0_EL1 characteristics are:

Purpose

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

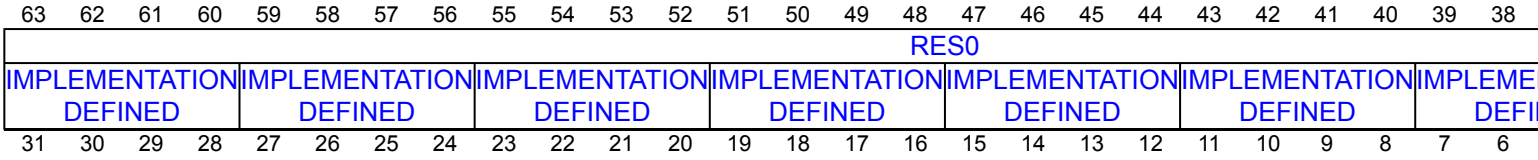
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64AFR0_EL1 are UNDEFINED.

Attributes

ID_AA64AFR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:28]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [27:24]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [23:20]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [19:16]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [15:12]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [11:8]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [7:4]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [3:0]

IMPLEMENTATION DEFINED.

Accessing ID_AA64AFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64AFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64AFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64AFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64AFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64AFR1_EL1, AArch64 Auxiliary Feature Register 1

The ID_AA64AFR1_EL1 characteristics are:

Purpose

Reserved for future expansion of information about the IMPLEMENTATION DEFINED features of the PE in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

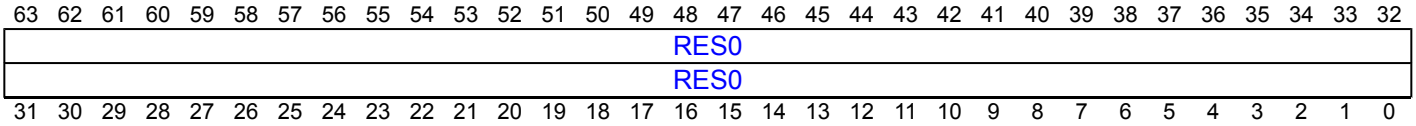
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64AFR1_EL1 are UNDEFINED.

Attributes

ID_AA64AFR1_EL1 is a 64-bit register.

Field descriptions



Bits [63:0]

Reserved, RES0.

Accessing ID_AA64AFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64AFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0101	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64AFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64AFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64AFR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0

The ID_AA64DFR0_EL1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

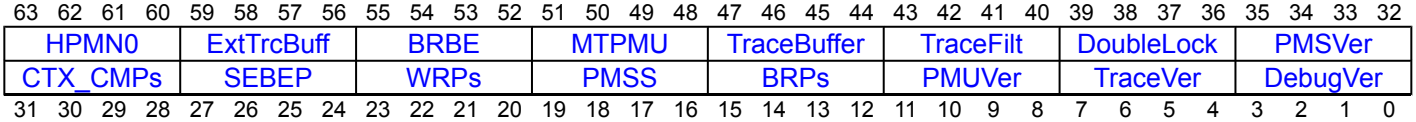
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64DFR0_EL1 are UNDEFINED.

The external register EDDFR gives information from this register.

Attributes

ID_AA64DFR0_EL1 is a 64-bit register.

Field descriptions



HPMN0, bits [63:60]

Zero PMU event counters for a Guest operating system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HPMN0	Meaning
0b0000	Setting MDCR_EL2.HPMN to zero has CONSTRAINED UNPREDICTABLE behavior.
0b0001	Setting MDCR_EL2.HPMN to zero has defined behavior.

All other values are reserved.

FEAT_HPMN0 implements the functionality identified by the value 0b0001.

From Armv8.8, in an implementation that includes FEAT_PMUv3, FEAT_FGT, and EL2, the value 0b0000 is not permitted.

Access to this field is **RO**.

ExtTrcBuff, bits [59:56]

Trace Buffer External Mode Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ExtTrcBuff	Meaning
0b0000	Trace Buffer External Mode not implemented.
0b0001	Trace Buffer External Mode implemented.

All other values are reserved.

FEAT_TRBE_EXT implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

BRBE, bits [55:52]

Branch Record Buffer Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BRBE	Meaning
0b0000	Branch Record Buffer Extension not implemented.
0b0001	Branch Record Buffer Extension implemented.
0b0010	As 0b0001, and adds support for branch recording at EL3.

All other values are reserved.

FEAT_BRBE implements the functionality identified by the value 0b0001.

FEAT_BRBEv1p1 implements the functionality identified by the value 0b0010.

From Armv9.3, if FEAT_BRBE is implemented, the value 0b0001 is not permitted.

Access to this field is **RO**.

MTPMU, bits [51:48]

Multi-threaded PMU extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTPMU	Meaning
0b0000	FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, it is IMPLEMENTATION DEFINED whether PMEVTYPER<n>_EL0.MT and PMEVTYPER<n>.MT are read/write or RES0.
0b0001	FEAT_MTPMU and FEAT_PMUv3 implemented. PMEVTYPER<n>_EL0.MT and PMEVTYPER<n>.MT are read/write. When FEAT_MTPMU is disabled, the Effective values of PMEVTYPER<n>_EL0.MT and PMEVTYPER<n>.MT are 0.
0b1111	FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, PMEVTYPER<n>_EL0.MT and PMEVTYPER<n>.MT are RES0.

All other values are reserved.

FEAT_MTPMU implements the functionality identified by the value 0b0001.

From Armv8.6, in an implementation that includes FEAT_PMUv3, the value 0b0000 is not permitted.

In an implementation that does not include FEAT_PMUv3, the value 0b0001 is not permitted.

Access to this field is **RO**.

TraceBuffer, bits [47:44]

Trace Buffer Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceBuffer	Meaning
0b0000	Trace Buffer Extension not implemented.
0b0001	Trace Buffer Extension implemented.
0b0010	As 0b0001, and adds: • If EL2 and FEAT_FGT are implemented, a fine-grained trap on the TSB CSYNC instruction. • If EL2 is implemented, an EL2 control to override TRBLIMITR_EL1.nVM. • The TRBE Profiling exception extension, FEAT_TRBE_EXC.

All other values are reserved.

FEAT_TRBE implements the functionality identified by the value 0b0001.

FEAT_TRBEv1p1 implements the functionality identified by the value 0b0010.

In any Armv9 implementation, if FEAT_ETE is implemented, the value 0b0000 is not permitted.

From Armv9.6, if FEAT_TRBE is implemented, the value 0b0001 is not permitted.

Access to this field is **RO**.

TraceFilt, bits [43:40]

Armv8.4 Self-hosted Trace Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceFilt	Meaning
0b0000	Armv8.4 Self-hosted Trace Extension not implemented.
0b0001	Armv8.4 Self-hosted Trace Extension implemented.

All other values are reserved.

FEAT_TRF implements the functionality identified by the value 0b0001.

From Armv8.4, if FEAT_ETMv4 is implemented, the value 0b0000 is not permitted.

If FEAT_ETE is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

DoubleLock, bits [39:36]

OS Double Lock implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DoubleLock	Meaning
0b0000	OS Double Lock implemented. OSDLR_EL1 is RW.
0b1111	OS Double Lock not implemented. OSDLR_EL1 is RAZ/WI.

All other values are reserved.

FEAT_DoubleLock implements the functionality identified by the value 0b0000.

In Armv8.0, the only permitted value is 0b0000.

If FEAT_Debugv8p2 is implemented and FEAT_DoPD is not implemented, the permitted values are 0b0000 and 0b1111.

If FEAT_DoPD is implemented, the only permitted value is 0b1111.

Access to this field is **RO**.

PMSVer, bits [35:32]

Statistical Profiling Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMSVer	Meaning
0b0000	Statistical Profiling Extension not implemented.
0b0001	Statistical Profiling Extension implemented.
0b0010	As 0b0001, and adds: • Support for the Events packet Alignment flag. • If FEAT_SVE is implemented, support for the Scalable Vector extensions to Statistical Profiling.
0b0011	As 0b0010, and adds: • Discard mode. • Extended event filtering, including the PMSNEVFR_EL1 System register. • Support for the OPTIONAL previous branch target Address packet. • If FEAT_PMUv3 is implemented, controls to freeze the PMU event counters after an SPE buffer management event occurs. • If FEAT_PMUv3 is implemented, the SAMPLE_FEED_BR, SAMPLE_FEED_EVENT, SAMPLE_FEED_LAT, SAMPLE_FEED_LD, SAMPLE_FEED_OP, and SAMPLE_FEED_ST PMU events.
0b0100	As 0b0011, and adds: • If FEAT_MOPS is implemented, Operation Type packet encodings for Memory Copy and Set operations. • If FEAT_MTE is implemented, Operation Type packet encodings for loads and stores of Allocation Tags.
0b0101	As 0b0100, and adds: • Support for the Events packet Level 2 Data cache access, Level 2 Data cache miss, Cached data modified, Recently fetched cache line, and Cache snoop flags. • Support for Data Source filtering.
0b0110	As 0b0101, and adds: • If EL2 and FEAT_FGT are implemented, a fine-grained trap on the PSB CSYNC instruction. • The SPE Profiling exception extension, FEAT_SPE_EXC. • The Statistical Profiling physical addressing mode extension, FEAT_SPE_nVM.

All other values are reserved.

FEAT_SPE implements the functionality identified by the value 0b0001.

FEAT_SPEv1p1 implements the functionality identified by the value 0b0010.

FEAT_SPEv1p2 implements the functionality identified by the value 0b0011.

FEAT_SPEv1p3 implements the functionality identified by the value 0b0100.

FEAT_SPEv1p4 implements the functionality identified by the value 0b0101.

FEAT_SPEv1p5 implements the functionality identified by the value 0b0110.

From Armv8.5, if FEAT_SPE is implemented, the value 0b0001 is not permitted.

From Armv8.7, if FEAT_SPE is implemented, the value 0b0010 is not permitted.

From Armv8.8, if FEAT_SPE is implemented, the value 0b0011 is not permitted.

From Armv8.9, if FEAT_SPE is implemented, the value 0b0100 is not permitted.

From Armv9.6, if FEAT_SPE is implemented, the value 0b0101 is not permitted.

Access to this field is **RO**.

CTX_CMPs, bits [31:28]

Number of context-aware breakpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CTX_CMPs	Meaning
0b0000..0b1111	The number of context-aware breakpoints, minus 1.

The value of this field is never greater than ID_AA64DFR0_EL1.BRPs.

If FEAT_Debugv8p9 is implemented and 16 or more context-aware breakpoints are implemented, then this field reads as 0b1111 and ID_AA64DFR1_EL1.CTX_CMPs indicates the number of context-aware breakpoints.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 breakpoints.

Access to this field is **RO**.

SEBEP, bits [27:24]

Synchronous-exception-based event profiling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEBEP	Meaning
0b0000	Synchronous-exception-based event profiling not implemented.
0b0001	Synchronous-exception-based event profiling implemented.

All other values are reserved.

FEAT_SEBEP implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

WRPs, bits [23:20]

Number of watchpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WRPs	Meaning
0b0001..0b1111	The number of watchpoints, minus 1.

If FEAT_Debugv8p9 is implemented and 16 or more watchpoints are implemented, then this field reads as 0b1111 and ID_AA64DFR1_EL1.WRPs indicates the number of watchpoints.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 watchpoints.

The value 0b0000 is reserved.

Access to this field is **RO**.

PMSS, bits [19:16]

PMU Snapshot extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMSS	Meaning
0b0000	PMU snapshot extension not implemented.
0b0001	PMU snapshot extension implemented.

All other values are reserved.

FEAT_PMUv3_SS implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

BRPs, bits [15:12]

Number of breakpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BRPs	Meaning
0b0001..0b1111	The number of breakpoints, minus 1.

If FEAT_Debugv8p9 is implemented and 16 or more breakpoints are implemented, then this field reads as 0b1111 and ID_AA64DFR1_EL1.BRPs indicates the number of breakpoints.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 breakpoints.

The value 0b0000 is reserved.

Access to this field is **RO**.

PMUVer, bits [11:8]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version'.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMUVer	Meaning
0b0000	Performance Monitors Extension not implemented.
0b0001	Performance Monitors Extension, PMUv3 implemented.
0b0100	PMUv3 for Armv8.1. As 0b0001, and adds support for: • Extended 16-bit PMEVTYPER<n>_EL0.evtCount field. • If EL2 is implemented, the MDCR_EL2.HPMD control.
0b0101	PMUv3 for Armv8.4. As 0b0100, and adds support for the PMMIR_EL1 register.
0b0110	PMUv3 for Armv8.5. As 0b0101, and adds support for: • 64-bit event counters. • If EL2 is implemented, the MDCR_EL2.HCCD control. • If EL3 is implemented, the MDCR_EL3.SCCD control.
0b0111	PMUv3 for Armv8.7. As 0b0110, and adds support for: • The PMCR_EL0.FZO and, if EL2 is implemented, MDCR_EL2.HPMFZO controls. • If EL3 is implemented, the MDCR_EL3.{MPMX,MCCD} controls.
0b1000	PMUv3 for Armv8.8. As 0b0111, and: • Extends the Common event number space to include 0x0040 to 0x00BF and 0x4040 to 0x40BF. • Removes the CONSTRAINED UNPREDICTABLE behaviors if a reserved or unimplemented PMU event number is selected.
0b1001	PMUv3 for Armv8.9. As 0b1000, and: • Updates the definitions of existing PMU events. • Adds support for the PMUSERENR_EL0.UEN control and the PMUACR_EL1 register. • Adds support for the EDECR.PME control.
0b1111	IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value for new implementations.

All other values are reserved.

FEAT_PMUv3 implements the functionality identified by the value 0b0001.

FEAT_PMUv3p1 implements the functionality identified by the value 0b0100.

FEAT_PMUv3p4 implements the functionality identified by the value 0b0101.

FEAT_PMUv3p5 implements the functionality identified by the value 0b0110.

FEAT_PMUv3p7 implements the functionality identified by the value 0b0111.

FEAT_PMUv3p8 implements the functionality identified by the value 0b1000.

FEAT_PMUv3p9 implements the functionality identified by the value 0b1001.

From Armv8.1, if FEAT_PMUv3 is implemented, the value 0b0001 is not permitted.

From Armv8.4, if FEAT_PMUv3 is implemented, the value 0b0100 is not permitted.

From Armv8.5, if FEAT_PMUv3 is implemented, the value 0b0101 is not permitted.

From Armv8.7, if FEAT_PMUv3 is implemented, the value 0b0110 is not permitted.

From Armv8.8, if FEAT_PMUv3 is implemented, the value 0b0111 is not permitted.

From Armv8.9, if FEAT_PMUv3 is implemented, the value 0b1000 is not permitted.

Access to this field is **RO**.

TraceVer, bits [7:4]

Trace support. Indicates whether System register interface to a trace unit is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceVer	Meaning
0b0000	Trace unit System registers not implemented.
0b0001	Trace unit System registers implemented.

All other values are reserved.

When trace unit System registers are implemented, see TRCIDR1 for tracing capabilities of the trace unit.

Access to this field is **RO**.

DebugVer, bits [3:0]

Debug architecture version. Indicates presence of Armv8 debug architecture.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DebugVer	Meaning
0b0110	Armv8.0 debug architecture.
0b0111	Armv8.1 debug architecture, FEAT_Debugv8p1.
0b1000	Armv8.2 debug architecture, FEAT_Debugv8p2.
0b1001	Armv8.4 debug architecture, FEAT_Debugv8p4.
0b1010	Armv8.8 debug architecture, FEAT_Debugv8p8.
0b1011	Armv8.9 debug architecture, FEAT_Debugv8p9.

All other values are reserved.

FEAT_Debugv8p1 implements the functionality identified by the value 0b0111.

FEAT_Debugv8p2 implements the functionality identified by the value 0b1000.

FEAT_Debugv8p4 implements the functionality identified by the value 0b1001.

FEAT_Debugv8p8 implements the functionality identified by the value 0b1010.

FEAT_Debugv8p9 implements the functionality identified by the value 0b1011.

From Armv8.1, when FEAT_Debugv8p1 is implemented the value 0b0110 is not permitted.

From Armv8.2, the values `0b0110` and `0b0111` are not permitted.

From Armv8.4, the value `0b1000` is not permitted.

From Armv8.8, the value `0b1001` is not permitted.

From Armv8.9, the value `0b1010` is not permitted.

Access to this field is **RO**.

Accessing ID_AA64DFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64DFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64DFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1

The ID_AA64DFR1_EL1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch64.

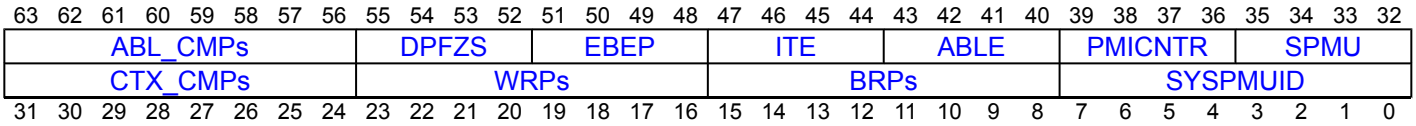
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64DFR1_EL1 are UNDEFINED.

Attributes

ID_AA64DFR1_EL1 is a 64-bit register.

Field descriptions



ABL_CMPs, bits [63:56]

When FEAT_ABLE is implemented:

Number of breakpoints that support address linking, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ABL_CMPs	Meaning
0x00..0x3F	Number of breakpoints that support address linking minus 1.

All other values are reserved.

The number of breakpoints that support address linking is never more than either the number of breakpoints or the number of watchpoints.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

DPFZS, bits [55:52]

Behavior of the cycle counter when event counting is frozen by a Statistical Profiling management event.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DPFZS	Meaning
0b0000	The cycle counter PMCCNTR_EL0 is never affected by PMCR_EL0.FZS.
0b0001	The cycle counter PMCCNTR_EL0 does not count when PMCR_EL0.DP is 1 and counting by event counters accessible to EL1 is frozen by the PMCR_EL0.FZS mechanism.

FEAT_SPE_DPFZS implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

EBEP, bits [51:48]

Exception-based event profiling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EBEP	Meaning
0b0000	Exception-based event profiling not implemented.
0b0001	Exception-based event profiling implemented.

All other values are reserved.

FEAT_EBEP implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

ITE, bits [47:44]

Instrumentation Trace Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ITE	Meaning
0b0000	Instrumentation Trace Extension not implemented.
0b0001	Instrumentation Trace Extension implemented.

All other values are reserved.

FEAT_ITE implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

ABLE, bits [43:40]

Address Breakpoint Linking Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ABLE	Meaning
0b0000	Address Breakpoint Linking Extension not implemented.
0b0001	Address Breakpoint Linking Extension implemented.

All other values are reserved.

FEAT_BWE implements the address range breakpoints and mismatch breakpoints part of the functionality identified by the value 0b0001.

FEAT_ABLE implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

PMICNTR, bits [39:36]

PMU fixed-function instruction counter.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMICNTR	Meaning
0b0000	PMU fixed-function instruction counter not implemented.
0b0001	PMU fixed-function instruction counter implemented.

All other values are reserved.

FEAT_PMUv3_ICNTR implements the functionality identified by the value 0b0001.

If FEAT_PMUv3 is not implemented, then the only permitted value is 0b0000.

Access to this field is **RO**.

SPMU, bits [35:32]

System PMU extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPMU	Meaning
0b0000	System PMU extension not implemented.
0b0001	System PMU extension implemented.
0b0010	As 0b0001, and adds support for SPMZR_EL0.

All other values are reserved.

FEAT_SPMU implements the functionality identified by the value 0b0001.

FEAT_SPMU2 implements the functionality identified by the value 0b0010.

From Armv9.5, the value 0b0001 is not permitted.

Access to this field is **RO**.

CTX_CMPs, bits [31:24]

Context-aware breakpoints.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CTX_CMPs	Meaning
0x00	ID_AA64DFR0_EL1.CTX_CMPs is the number of context-aware breakpoints, minus 1.
0x01..0x3F	Number of context-aware breakpoints minus 1.

All other values are reserved.

The value of this field is never greater than ID_AA64DFR1_EL1.BRPs.

Access to this field is **RO**.

WRPs, bits [23:16]

Watchpoints.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WRPs	Meaning
0x00	ID_AA64DFR0_EL1.WRPs is the number of watchpoints, minus 1.
0x01..0x3F	Number of watchpoints minus 1.

All other values are reserved.

Access to this field is **RO**.

BRPs, bits [15:8]

Breakpoints.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BRPs	Meaning
0x00	ID_AA64DFR0_EL1.BRPs is the number of breakpoints, minus 1.
0x01..0x3F	Number of breakpoints minus 1.

All other values are reserved.

Access to this field is **RO**.

SYSPMUID, bits [7:0]

When FEAT_SPMU is implemented:

System PMU ID. Indicates the largest value that can be written to SPMSELR_EL0.SYSPMUSEL.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYSPMUID	Meaning
0x00..0x1F	The largest supported value that can be written to SPMSELR_EL0.SYSPMUSEL.

All other values are reserved.

Since System PMUs might not be contiguously accessible, this field does not necessarily indicate the total number of accessible System PMUs.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing ID_AA64DFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64DFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0101	0b001

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64DFR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64DFR2_EL1, AArch64 Debug Feature Register 2

The ID_AA64DFR2_EL1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch64.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64DFR2_EL1 are UNDEFINED.

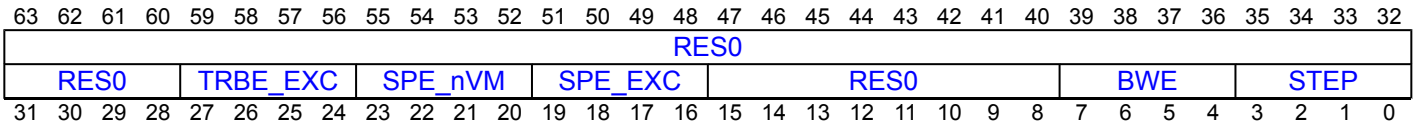
Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64DFR2_EL1 is a 64-bit register.

Field descriptions



Bits [63:28]

Reserved, RES0.

TRBE_EXC, bits [27:24]

TRBE Profiling exception extension. Describes support for reporting trace buffer management events as TRBE Profiling exceptions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRBE_EXC	Meaning
0b0000	TRBE Profiling exception not implemented.
0b0001	TRBE Profiling exception implemented.

All other values are reserved.

FEAT_TRBE_EXC implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

SPE_nVM, bits [23:20]

Profiling Buffer physical address mode supported. Describes support for defining the Profiling Buffer using physical addresses or intermediate physical addresses.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPE_nVM	Meaning
0b0000	Profiling Buffer physical address mode not implemented.
0b0001	Profiling Buffer physical address mode implemented.

All other values are reserved.

FEAT_SPE_nVM implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

SPE_EXC, bits [19:16]

SPE Profiling exception extension. Describes support for reporting Profiling Buffer management events as SPE Profiling exceptions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPE_EXC	Meaning
0b0000	SPE Profiling exception not implemented.
0b0001	SPE Profiling exception implemented.

All other values are reserved.

FEAT_SPE_EXC implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

Bits [15:8]

Reserved, RES0.

BWE, bits [7:4]

Breakpoints and watchpoint enhancements.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BWE	Meaning
0b0000	This field does not indicate whether DBGBCR<n>_EL1.MASK and address mismatch breakpoints are implemented.
0b0001	DBGBCR<n>_EL1.MASK and address mismatch breakpoints are implemented.
0b0010	As 0b0001, and address mismatch watchpoints are implemented.

All other values are reserved.

FEAT_BWE implements the functionality identified by the value 0b0001.

FEAT_BWE2 implements the functionality identified by the value 0b0010.

When this field is 0b0000, ID_AA64DFR1_EL1.ABLE might indicate the presence of support for DBGBCR<n>_EL1.MASK and address mismatch breakpoints.

From Armv9.5, the value 0b0001 is not permitted.

Access to this field is **RO**.

STEP, bits [3:0]

Enhanced Software Step Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

STEP	Meaning
0b0000	Execution from MDSTEPOP_EL1 is not supported for Software Step.
0b0001	Execution from MDSTEPOP_EL1 is supported for Software Step.

All other values are reserved.

FEAT_STEP2 implements the functionality identified by the value `0b0001`.

Access to this field is **RO**.

Accessing ID_AA64DFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

`MRS <Xt>, ID_AA64DFR2_EL1`

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0101	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64DFR2_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_AA64DFR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64DFR2_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64FPFR0_EL1, AArch64 Floating-point Feature Register 0

The ID_AA64FPFR0_EL1 characteristics are:

Purpose

Provides information about floating-point formats and instructions implemented in AArch64 state.

The fields in this register do not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64SMFR0_EL1 and ID_AA64FPFR0_EL1.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64FPFR0_EL1 are UNDEFINED.

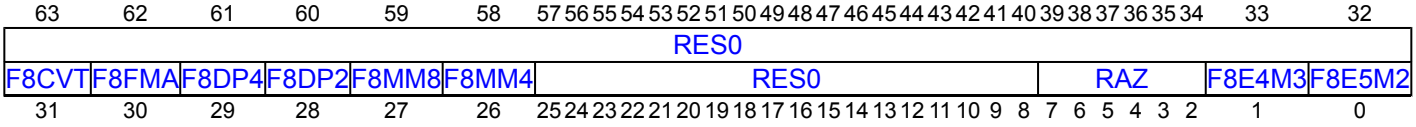
Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64FPFR0_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

F8CVT, bit [31]

Indicates support for the following instructions:

- The Advanced SIMD floating-point scaling instruction `FSCALE`.
- The Advanced SIMD FP8 convert instructions `BF1CVTL`, `BF1CVTL2`, `BF2CVTL`, `BF2CVTL2`, `F1CVTL`, `F1CVTL2`, `F2CVTL`, `F2CVTL2`, `FCVTN`, and `FCVTN2`.
- When FEAT_SVE2 or FEAT_SME2 is implemented, the SVE2 FP8 convert instructions `BF1CVT`, `BF1CVTLT`, `BF2CVT`, `BF2CVTLT`, `F1CVT`, `F1CVTLT`, `F2CVT`, `F2CVTLT`, `BFCVTN`, `FCVTN`, `FCVTNB`, and `FCVTNT`.
- When FEAT_SME2 is implemented, the SME2 multi-vector floating-point scaling instruction `FSCALE` and the SME2 FP8 convert instructions `BF1CVT`, `BF1CVTL`, `BF2CVT`, `BF2CVTL`, `F1CVT`, `F1CVTL`, `F2CVT`, `F2CVTL`, `BFCVT`, `FCVT`, and `FCVTN`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8CVT	Meaning
`0b0`	The specified instructions are not implemented.
`0b1`	The specified instructions are implemented.

FEAT_FP8 implements the functionality identified by the value 1.

Access to this field is **RO**.

F8FMA, bit [30]

Indicates support for the following instructions:

- The Advanced SIMD FP8 to single-precision and half-precision multiply-accumulate instructions `FMLALB`, `FMLALT`, `FMLALLBB`, `FMLALLBT`, `FMLALLTB`, and `FMLALLTT`.
- When FEAT_SVE2 is implemented and the PE is not in Streaming SVE mode, the SVE2 FP8 to single-precision and half-precision multiply-accumulate instructions `FMLALB`, `FMLALT`, `FMLALLBB`, `FMLALLBT`, `FMLALLTB`, and `FMLALLTT`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8FMA	Meaning
`0b0`	The specified instructions are not implemented.
`0b1`	The specified instructions are implemented.

FEAT_FP8FMA implements the functionality identified by the value 1.

Access to this field is **RO**.

F8DP4, bit [29]

Indicates support for the following instructions:

- Advanced SIMD FP8 to single-precision 4-way dot product `FDOT` (4-way) instructions.
- When FEAT_SVE2 is implemented and the PE is not in Streaming SVE mode, SVE FP8 to single-precision 4-way dot product `FDOT` (4-way) instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8DP4	Meaning
`0b0`	The specified instructions are not implemented by this feature.
`0b1`	The specified instructions are implemented.

Note

Other features may implement some of the specified instructions.

All other values are reserved.

FEAT_FP8DOT4 implements the functionality identified by the value 1.

Access to this field is **RO**.

F8DP2, bit [28]

Indicates support for the following instructions:

- Advanced SIMD FP8 to half-precision 2-way dot product `FDOT` (2-way) instructions.
- When FEAT_SVE2 is implemented and the PE is not in Streaming SVE mode, SVE FP8 to half-precision 2-way dot product `FDOT` (2-way) instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8DP2	Meaning
`0b0`	The specified instructions are not implemented by this feature.
`0b1`	The specified instructions are implemented.

Note

Other features may implement some of the specified instructions.

FEAT_FP8DOT2 implements the functionality identified by the value 1.

Access to this field is **RO**.

F8MM8, bit [27]

Indicates support for the following instructions:

- Advanced SIMD FP8 to single-precision matrix multiply `FMMLA` (8-way, FP8 to FP32) instruction.
- If FEAT_SVE2 is implemented, SVE FP8 to single-precision matrix multiply `FMMLA` (widening, FP8 to FP32) instruction is implemented when the PE is not in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8MM8	Meaning
`0b0`	Advanced SIMD and SVE FP8 to single-precision matrix multiply instructions are not implemented by this feature.
`0b1`	The specified Advanced SIMD and SVE FP8 to single-precision matrix multiply instructions are implemented.

FEAT_F8F32MM implements the functionality identified by the value `0b0001`

Access to this field is **RO**.

F8MM4, bit [26]

Indicates support for the following instructions:

- Advanced SIMD FP8 to half-precision matrix multiply `FMMLA` (4-way, FP8 to FP16) instruction.
- If FEAT_SVE2 is implemented, SVE FP8 to half-precision matrix multiply `FMMLA` (widening, FP8 to FP16) instruction is implemented when the PE is not in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8MM4	Meaning
`0b0`	Advanced SIMD and SVE FP8 to half-precision matrix multiply instructions are not implemented by this feature.
`0b1`	The specified Advanced SIMD and SVE FP8 to half-precision matrix multiply instructions are implemented.

FEAT_F8F16MM implements the functionality identified by the value `0b0001`

Access to this field is **RO**.

Bits [25:8]

Reserved, RES0.

Bits [7:2]

Reserved for data formats 2 to 7.

Reserved, RAZ.

F8E4M3, bit [1]

Indicates support for OFP8 E4M3 format and behavior for FP8 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8E4M3	Meaning
`0b0`	OFP8 E4M3 format is not supported.
`0b1`	OFP8 E4M3 format is supported.

If FEAT_FP8 is implemented, the only permitted value is 1.

Otherwise, the only permitted value is 0.

For more information on OFP8 formats, see the Open Compute Project, OCP 8-bit Floating Point Specification (OFP8).

Access to this field is **RO**.

F8E5M2, bit [0]

Indicates support for OFP8 E5M2 format and behavior for FP8 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8E5M2	Meaning
0b0	OFP8 E5M2 format is not supported.
0b1	OFP8 E5M2 format is supported.

If FEAT_FP8 is implemented, the only permitted value is 1.

Otherwise, the only permitted value is 0.

For more information on OFP8 formats, see the Open Compute Project, OCP 8-bit Floating Point Specification (OFP8).

Access to this field is **RO**.

Accessing ID_AA64FPFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64FPFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64FPFR0_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_AA64FPFR0_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64FPFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64FPFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64FPFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0

The ID_AA64ISAR0_EL1 characteristics are:

Purpose

Provides information about the instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64ISAR0_EL1 are UNDEFINED.

Attributes

ID_AA64ISAR0_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RNDR				TLB				TS				FHM				DP				SM4				SM3				SHA3			
RDM				TME				Atomic				CRC32				SHA2				SHA1				AES				RES0			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

RNDR, bits [63:60]

Indicates support for Random Number instructions in AArch64 state.

When FEAT_RNG_TRAP is implemented, the value returned by a direct read of ID_AA64ISAR0_EL1.RNDR is further controlled by the value of SCR_EL3.TRNDR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RNDR	Meaning
0b0000	No Random Number instructions are implemented.
0b0001	RNDR and RNDRRS registers are implemented.

All other values are reserved.

FEAT_RNG implements the functionality identified by the value 0b0001.

From Armv8.5, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

TLB, bits [59:56]

Indicates support for Outer Shareable and TLB range maintenance instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TLB	Meaning
0b0000	Outer Shareable and TLB range maintenance instructions are not implemented.
0b0001	Outer Shareable TLB maintenance instructions are implemented.
0b0010	Outer Shareable and TLB range maintenance instructions are implemented.

All other values are reserved.

FEAT_TLBIOS implements the functionality identified by the values 0b0001 and 0b0010.

FEAT_TLBIRANGE implements the functionality identified by the value 0b0010.

From Armv8.4, the values 0b0000 and 0b0001 are not permitted.

Access to this field is **RO**.

TS, bits [55:52]

Indicates support for flag manipulation instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TS	Meaning
0b0000	No flag manipulation instructions are implemented.
0b0001	CFINV, RMIF, SETF16, and SETF8 instructions are implemented.
0b0010	CFINV, RMIF, SETF16, SETF8, AXFLAG, and XAFLAG instructions are implemented.

All other values are reserved.

FEAT_FlagM implements the functionality identified by the value 0b0001.

FEAT_FlagM2 implements the functionality identified by the value 0b0010.

In Armv8.4, the value 0b0000 is not permitted.

From Armv8.5, the value 0b0001 is not permitted.

Access to this field is **RO**.

FHM, bits [51:48]

Indicates support for the Advanced SIMD half-precision to single-precision multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FHM	Meaning
0b0000	The Advanced SIMD half-precision to single-precision multiply instructions are not implemented.
0b0001	FMLAL and FMLSL instructions are implemented.

All other values are reserved.

FEAT_FHM implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

DP, bits [47:44]

Indicates support for Dot Product instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DP	Meaning
0b0000	No Dot Product instructions implemented.
0b0001	UDOT and SDOT instructions implemented.

All other values are reserved.

FEAT_DotProd implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

SM4, bits [43:40]

Indicates support for SM4 instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SM4	Meaning
0b0000	No SM4 instructions implemented.
0b0001	SM4E and SM4EKEY instructions implemented.

All other values are reserved.

If FEAT_SM4 is not implemented, the value 0b0001 is reserved.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

This field must have the same value as ID_AA64ISAR0_EL1.SM3.

Access to this field is **RO**.

SM3, bits [39:36]

Indicates support for the following SM3 instructions SM3SS1, SM3TT1A, SM3TT1B, SM3TT2A, SM3TT2B, SM3PARTW1, and SM3PARTW2 in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SM3	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

If FEAT_SM3 is not implemented, the value 0b0001 is reserved.

FEAT_SM3 implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

This field must have the same value as ID_AA64ISAR0_EL1.SM4.

Access to this field is **RO**.

SHA3, bits [35:32]

Indicates support for the following SHA3 instructions EOR3, RAX1, XAR, and BCAX in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA3	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

If FEAT_SHA3 is not implemented, the value 0b0001 is reserved.

FEAT_SHA3 implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR0_EL1.SHA1 is 0b0000, this field must have the value 0b0000.

If the value of this field is 0b0001, ID_AA64ISAR0_EL1.SHA2 must have the value 0b0010.

Access to this field is **RO**.

RDM, bits [31:28]

Indicates support for SQRDMLAH and SQRDMLSH instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RDM	Meaning
0b0000	No RDMA instructions implemented.
0b0001	SQRDMLAH and SQRDMLSH instructions implemented.

All other values are reserved.

FEAT_RDM implements the functionality identified by the value 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

TME, bits [27:24]

Indicates support for the following TME instructions TCANCEL, TCOMMIT, TSTART, and TTEST.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TME	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_TME is implemented.
  - PSTATE.EL IN {EL2, EL1}.
  - SCR_EL3.TME == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_TME is implemented.
  - PSTATE.EL == EL1.
  - EL2Enabled().
  - HCR_EL2.TME == 0.
- Otherwise, access to this field is **RO**.

Atomic, bits [23:20]

Indicates support for Atomic instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Atomic	Meaning
0b0000	No Atomic instructions implemented.
0b0010	LDADD, LDCLR, LDEOR, LDSET, LDSMAX, LDSMIN, LDUMAX, LDUMIN, CAS, CASP, and SWP instructions implemented.
0b0011	As for 0b0010, plus 128-bit instructions LDCLRP, LDSETP, and SWPP.

All other values are reserved.

FEAT_LSE implements the functionality identified by the value 0b0010.

FEAT_LSE128 implements the functionality identified by the value 0b0011.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

CRC32, bits [19:16]

Indicates support for the following CRC32 instructions `CRC32B`, `CRC32H`, `CRC32W`, `CRC32X`, `CRC32CB`, `CRC32CH`, `CRC32CW`, and `CRC32CX` in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CRC32	Meaning
`0b0000`	The specified instructions are not implemented.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_CRC32 implements the functionality identified by the value `0b0001`.

From Armv8.1, the value `0b0000` is not permitted.

Access to this field is **RO**.

SHA2, bits [15:12]

Indicates support for SHA2 instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA2	Meaning
`0b0000`	No SHA2 instructions implemented.
`0b0001`	Implements instructions: `SHA256H`, `SHA256H2`, `SHA256SU0`, and `SHA256SU1`.
`0b0010`	Implements instructions: • `SHA256H`, `SHA256H2`, `SHA256SU0`, and `SHA256SU1`. • `SHA512H`, `SHA512H2`, `SHA512SU0`, and `SHA512SU1`.

All other values are reserved.

FEAT_SHA256 implements the functionality identified by the value `0b0001`.

FEAT_SHA512 implements the functionality identified by the value `0b0010`.

If the value of ID_AA64ISAR0_EL1.SHA1 is `0b0000`, this field must have the value `0b0000`.

If the value of this field is `0b0010`, ID_AA64ISAR0_EL1.SHA3 must have the value `0b0001`.

Access to this field is **RO**.

SHA1, bits [11:8]

Indicates support for the following SHA1 instructions `SHA1C`, `SHA1P`, `SHA1M`, `SHA1H`, `SHA1SU0`, and `SHA1SU1` in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA1	Meaning
`0b0000`	The specified instructions are not implemented.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_SHA1 implements the functionality identified by the value `0b0001`.

If the value of ID_AA64ISAR0_EL1.SHA2 is `0b0000`, this field must have the value `0b0000`.

Access to this field is **RO**.

AES, bits [7:4]

Indicates support for AES instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AES	Meaning
0b0000	No AES instructions implemented.
0b0001	AESE, AESD, AESMC, and AESIMC instructions implemented.
0b0010	As for 0b0001, plus PMULL and PMULL2 instructions operating on 64-bit source elements.

FEAT_AES implements the functionality identified by the value 0b0001.

FEAT_PMULL implements the functionality identified by the value 0b0010.

All other values are reserved.

Access to this field is **RO**.

Bits [3:0]

Reserved, RES0.

Accessing ID_AA64ISAR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64ISAR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64ISAR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1

The ID_AA64ISAR1_EL1 characteristics are:

Purpose

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

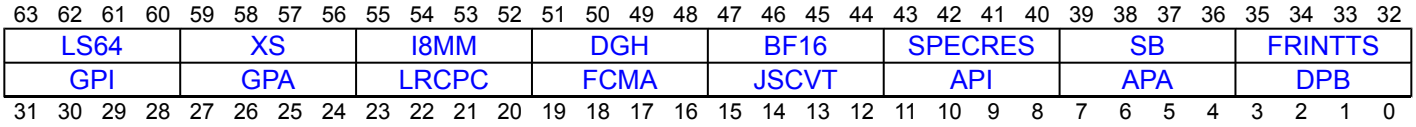
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64ISAR1_EL1 are UNDEFINED.

Attributes

ID_AA64ISAR1_EL1 is a 64-bit register.

Field descriptions



LS64, bits [63:60]

Indicates support for `LD64B` and `ST64B*` instructions, and the ACCDATA_EL1 register.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LS64	Meaning
`0b0000`	No `LD64B` or `ST64B` instructions are supported.
`0b0001`	The `LD64B` and `ST64B` instructions are supported.
`0b0010`	As `0b0001`, and adds the `ST64BV` instruction and traps.
`0b0011`	As `0b0010`, and adds the `ST64BV0` instruction, ACCDATA_EL1 register, and traps.
`0b0100`	As `0b0011`, and adds support for atomic accesses to Write-back Cacheable, Shareable memory using one of the following: • `LD64B` and `ST64B` instructions. • SIMD&FP instructions that load or store a pair of 128-bit registers by generating 32-byte single-copy atomic accesses.

All other values are reserved.

FEAT_LS64 implements the functionality identified by `0b0001`.

FEAT_LS64_V implements the functionality identified by `0b0010`.

FEAT_LS64_ACCDATA implements the functionality identified by `0b0011`.

FEAT_LS64WB implements the functionality identified by `0b0100`.

Access to this field is **RO**.

XS, bits [59:56]

Indicates support for the XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the HCRX_EL2.{FGTnXS, FnXS} fields in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

XS	Meaning
0b0000	The XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the HCRX_EL2.{FGTnXS, FnXS} fields are not supported.
0b0001	The XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the HCRX_EL2.{FGTnXS, FnXS} fields are supported.

All other values are reserved.

FEAT_XS implements the functionality identified by 0b0001.

From Armv8.7, the value 0b0000 is not permitted.

Access to this field is **RO**.

I8MM, bits [55:52]

Indicates support for the following Advanced SIMD Int8 matrix multiplication instructions SMMLA, SUDOT, UMMLA, USMMLA, and USDOT in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I8MM	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_I8MM implements the functionality identified by 0b0001.

When Advanced SIMD and SVE are both implemented, this field must return the same value as ID_AA64ZFR0_EL1.I8MM.

From Armv8.6, the value 0b0000 is not permitted.

Access to this field is **RO**.

DGH, bits [51:48]

Indicates support for the Data Gathering Hint instruction.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DGH	Meaning
0b0000	Data Gathering Hint is not implemented.
0b0001	Data Gathering Hint is implemented.

All other values are reserved.

FEAT_DGH implements the functionality identified by 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

If the DGH instruction has no effect in preventing the merging of memory accesses, the value of this field is 0b0000.

Access to this field is **RO**.

BF16, bits [47:44]

Indicates support for Advanced SIMD and floating-point BFloat16 instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BF16	Meaning
0b0000	BFloat16 instructions are not implemented.
0b0001	BFCVT, BFCVTN, BFCVTN2, BFDOT, BFMLALB, BFMLALT, and BFMMLA instructions are implemented.
0b0010	As 0b0001, but the FPCR.EBF field is also supported.

All other values are reserved.

FEAT_BF16 implements the functionality identified by 0b0001.

FEAT_EBF16 implements the functionality identified by 0b0010.

When FEAT_SVE or FEAT_SME is implemented, this field must return the same value as ID_AA64ZFR0_EL1.BF16.

From Armv8.6 and Armv9.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

SPECRES, bits [43:40]

Indicates support for prediction invalidation instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPECRES	Meaning
0b0000	Prediction invalidation instructions are not implemented.
0b0001	CFP RCTX, DVP RCTX and CPP RCTX instructions are implemented.
0b0010	As 0b0001, and COSP RCTX instruction is implemented.

All other values are reserved.

FEAT_SPECRES implements the functionality identified by 0b0001.

FEAT_SPECRES2 implements the functionality identified by 0b0010.

From Armv8.5, the value 0b0000 is not permitted.

From Armv8.9, the value 0b0001 is not permitted.

Access to this field is **RO**.

SB, bits [39:36]

Indicates support for SB instruction in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SB	Meaning
0b0000	SB instruction is not implemented.
0b0001	SB instruction is implemented.

All other values are reserved.

FEAT_SB implements the functionality identified by 0b0001.

From Armv8.5, the value 0b0000 is not permitted.

Access to this field is **RO**.

FRINTTS, bits [35:32]

Indicates support for FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FRINTTS	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_FRINTTS implements the functionality identified by 0b0001.

From Armv8.5, the value 0b0000 is not permitted.

Access to this field is **RO**.

GPI, bits [31:28]

Indicates support for an IMPLEMENTATION DEFINED algorithm is implemented in the PE for generic code authentication in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GPI	Meaning
0b0000	Generic Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented.
0b0001	Generic Authentication using an IMPLEMENTATION DEFINED algorithm is implemented. This includes the PACGA instruction.

All other values are reserved.

When this field is nonzero, FEAT_PACIMP is implemented.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPA is nonzero, or the value of ID_AA64ISAR2_EL1.GPA3 is nonzero, this field must have the value 0b0000.

Access to this field is **RO**.

GPA, bits [27:24]

Indicates whether the QARMA5 algorithm is implemented in the PE for generic code authentication in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GPA	Meaning
0b0000	Generic Authentication using the QARMA5 algorithm is not implemented.
0b0001	Generic Authentication using the QARMA5 algorithm is implemented. This includes the PACGA instruction.

All other values are reserved.

FEAT_PACQARMA5 implements the functionality identified by 0b0001.

When this field is nonzero, FEAT_PACQARMA5 is implemented.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPI is nonzero, or the value of ID_AA64ISAR2_EL1.GPA3 is nonzero, this field must have the value 0b0000.

Access to this field is **RO**.

LRCPC, bits [23:20]

Indicates support for weaker release consistency, RCpc, based model.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LRCPC	Meaning
0b0000	RCpc instructions are not implemented.
0b0001	The no offset LDAPR, LDAPRB, and LDAPRH instructions are implemented.
0b0010	As 0b0001, and the LDAPR (unscaled immediate) and STLR (unscaled immediate) instructions are implemented.
0b0011	As 0b0010, and the post-index LDAPR, LDIAPP, STILP, and pre-index STLR instructions are implemented. If Advanced SIMD and floating-point is implemented, then the LDAPUR (SIMD&FP), LDAP1 (SIMD&FP), STLUR (SIMD&FP), and STL1 (SIMD&FP) instructions are implemented in Advanced SIMD and floating-point.

All other values are reserved.

FEAT_LRCPC implements the functionality identified by the value 0b0001.

FEAT_LRCPC2 implements the functionality identified by the value 0b0010.

FEAT_LRCPC3 implements the functionality identified by the value 0b0011.

From Armv8.3, the value 0b0000 is not permitted.

From Armv8.4, the value 0b0001 is not permitted.

Access to this field is **RO**.

FCMA, bits [19:16]

Indicates support for complex number addition and multiplication, where numbers are stored in vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FCMA	Meaning
0b0000	The FCMLA and FCADD instructions are not implemented.
0b0001	The FCMLA and FCADD instructions are implemented.

All other values are reserved.

FEAT_FCMA implements the functionality identified by the value 0b0001.

From Armv8.3, if Advanced SIMD or floating-point is implemented, the value 0b0000 is not permitted.

From Armv8.3, if Advanced SIMD or floating-point is not implemented, the only permitted value is 0b0000.

Access to this field is **RO**.

JSCVT, bits [15:12]

Indicates support for JavaScript conversion from double-precision floating-point values to integers in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

JSCVT	Meaning
0b0000	The FJCVTZS instruction is not implemented.
0b0001	The FJCVTZS instruction is implemented.

All other values are reserved.

FEAT_JSCVT implements the functionality identified by 0b0001.

From Armv8.3, if Advanced SIMD or floating-point is implemented, the value 0b0000 is not permitted.

From Armv8.3, if Advanced SIMD or floating-point is not implemented, the only permitted value is 0b0000.

Access to this field is **RO**.

API, bits [11:8]

Indicates whether an IMPLEMENTATION DEFINED algorithm is implemented in the PE for address authentication, in AArch64 state. This applies to all Pointer Authentication instructions other than the `PACGA` instruction.

The value of this field is an IMPLEMENTATION DEFINED choice of:

API	Meaning
`0b0000`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented.
`0b0001`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, FEAT_EPAC and FEAT_PAuth2 are not implemented.
`0b0010`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, FEAT_EPAC is implemented, and FEAT_PAuth2 is not implemented.
`0b0011`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, FEAT_EPAC is not implemented, and FEAT_PAuth2 is implemented.
`0b0100`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2 and FEAT_FPAC are implemented.
`0b0101`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, and FEAT_FPACCOMBINE are implemented.
`0b0110`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, including instructions that allow signing of LR using SP and PC as diversifiers, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, FEAT_FPACCOMBINE, and FEAT_PAuth_LR are implemented.

All other values are reserved.

FEAT_PAuth implements the functionality identified by `0b0001`.

FEAT_EPAC implements the functionality identified by `0b0010`.

FEAT_PAuth2 implements the functionality identified by `0b0011`.

FEAT_FPAC implements the functionality identified by `0b0100`.

FEAT_FPACCOMBINE implements the functionality identified by `0b0101`.

FEAT_PAuth_LR implements the functionality identified by `0b0110`.

When this field is nonzero, FEAT_PACIMP is implemented.

In Armv8.3, the permitted values are `0b0000`, `0b0001`, `0b0010`, `0b0011`, `0b0100`, and `0b0101`.

From Armv8.6, the permitted values are `0b0000`, `0b0011`, `0b0100`, and `0b0101`.

From Armv9.5, the permitted values are `0b0000`, `0b0011`, `0b0100`, `0b0101`, and `0b0110`.

If the value of ID_AA64ISAR1_EL1.APA is nonzero, or the value of ID_AA64ISAR2_EL1.APA3 is nonzero, this field must have the value `0b0000`.

Access to this field is **RO**.

APA, bits [7:4]

Indicates whether the QARMA5 algorithm is implemented in the PE for address authentication, in AArch64 state. This applies to all Pointer Authentication instructions other than the `PACGA` instruction.

The value of this field is an IMPLEMENTATION DEFINED choice of:

APA	Meaning
0b0000	Address Authentication using the QARMA5 algorithm is not implemented.
0b0001	Address Authentication using the QARMA5 algorithm is implemented, FEAT_EPAC and FEAT_PAuth2 are not implemented.
0b0010	Address Authentication using the QARMA5 algorithm is implemented, FEAT_EPAC is implemented, and FEAT_PAuth2 is not implemented.
0b0011	Address Authentication using the QARMA5 algorithm is implemented, FEAT_EPAC is not implemented, and FEAT_PAuth2 is implemented.
0b0100	Address Authentication using the QARMA5 algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2 and FEAT_FPAC are implemented.
0b0101	Address Authentication using the QARMA5 algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, and FEAT_FPACCOMBINE are implemented.
0b0110	Address Authentication using the QARMA5 algorithm is implemented, including instructions that allow signing of LR using SP and PC as diversifiers, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, FEAT_FPACCOMBINE, and FEAT_PAuth_LR are implemented.

All other values are reserved.

FEAT_PAuth implements the functionality identified by 0b0001.

FEAT_EPAC implements the functionality identified by 0b0010.

FEAT_PAuth2 implements the functionality identified by 0b0011.

FEAT_FPAC implements the functionality identified by 0b0100.

FEAT_FPACCOMBINE implements the functionality identified by 0b0101.

FEAT_PAuth_LR implements the functionality identified by 0b0110.

When this field is nonzero, FEAT_PACQARMA5 is implemented.

In Armv8.3, the permitted values are 0b0000, 0b0001, 0b0010, 0b0011, 0b0100, and 0b0101.

From Armv8.6, the permitted values are 0b0000, 0b0011, 0b0100, and 0b0101.

From Armv9.5, the permitted values are 0b0000, 0b0011, 0b0100, 0b0101, and 0b0110.

If the value of ID_AA64ISAR1_EL1.API is nonzero, or the value of ID_AA64ISAR2_EL1.APA3 is nonzero, this field must have the value 0b0000.

Access to this field is **RO**.

DPB, bits [3:0]

Data Persistence writeback. Indicates support for the DC CVAP and DC CVADP instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DPB	Meaning
0b0000	DC CVAP not supported.
0b0001	DC CVAP supported.
0b0010	DC CVAP and DC CVADP supported.

All other values are reserved.

FEAT_DPB implements the functionality identified by the value 0b0001.

FEAT_DPB2 implements the functionality identified by the value 0b0010.

From Armv8.2, the value 0b0000 is not permitted.

From Armv8.5, the value 0b0001 is not permitted.

Access to this field is **RO**.

Accessing ID_AA64ISAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64ISAR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0110	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64ISAR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64ISAR2_EL1, AArch64 Instruction Set Attribute Register 2

The ID_AA64ISAR2_EL1 characteristics are:

Purpose

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64ISAR2_EL1 are UNDEFINED.

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64ISAR2_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
ATS1A				LUT				CSSC				RPRFM				PCDPHINT				PRFMSLC				SYSINSTR_128				SYSREG_128			
CLRBHB				PAC_frac				BC				MOPS				APA3				GPA3				RPRES				WFxT			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ATS1A, bits [63:60]

Indicates support for address translation instructions, which perform stage 1 address translation for the given virtual address without checking for stage 1 permissions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ATS1A	Meaning
0b0000	Address Translate Stage 1 instructions without Permissions Checks are not implemented
0b0001	Address Translate Stage 1 instructions without Permissions Checks are implemented.

All other values are reserved.

Access to this field is **RO**.

LUT, bits [59:56]

Indicates support for:

- Advanced SIMD lookup table instructions with 2-bit and 4-bit indices.
- If FEAT_SVE2 or FEAT_SME2 is implemented, SVE lookup table instructions with 2-bit and 4-bit indices.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LUT	Meaning
0b0000	The specified instructions are not implemented.
0b0001	Lookup table instructions with 2-bit indices LUTI2 and 4-bit indices LUTI4 are implemented.

All other values are reserved.

FEAT_LUT implements the functionality identified by the value 0b0001.

From Armv9.5, if FEAT_AdvSIMD is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

CSSC, bits [55:52]

Indicates support for common short sequence compression instructions.

If FEAT_CMPBR is implemented, indicates support for compare and branch instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSSC	Meaning
0b0000	Common short sequence compression instructions are not implemented.
0b0001	The following common short sequence compression instructions are implemented: • 32-bit and 64-bit ABS, CNT, CTZ • 32-bit SMAX, UMAX, SMIN, UMIN (immediate). • 64-bit SMAX, UMAX, SMIN, UMIN (immediate). • 32-bit SMAX, UMAX, SMIN, UMIN (register). • 64-bit SMAX, UMAX, SMIN, UMIN (register).
0b0010	As 0b0001, and adds compare and branch instructions: • CBB<cc>. • CB<cc> (immediate), CB<cc> (register). • CBH<cc>.

All other values are reserved.

FEAT_CSSC implements the functionality identified by the value 0b0001.

FEAT_CMPBR implements the functionality identified by the value 0b0010.

From Armv9.4, the value 0b0000 is not permitted.

From Armv9.6, the value 0b0001 is not permitted.

Access to this field is **RO**.

RPRFM, bits [51:48]

RPRFM hint instruction.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RPRFM	Meaning
0b0000	RPRFM hint instruction is not implemented and is treated as a NOP.
0b0001	RPRFM hint instruction is implemented.

All other values are reserved.

FEAT_RPRFM implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

PCDPHINT, bits [47:44]

Indicates support for producer-consumer data placement hints.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PCDPHINT	Meaning
0b0000	The STSHH and PRFM IR hint instructions are not implemented.
0b0001	The STSHH and PRFM IR hint instructions are implemented.

FEAT_PCDPHINT implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

PRFMSLC, bits [43:40]

Indicates whether the PRFM and PRFUM instructions support a system level cache option.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PRFMSLC	Meaning
0b0000	The PRFM and PRFUM instructions do not support the SLC target.
0b0001	The PRFM and PRFUM instructions support the SLC target.

All other values are reserved.

FEAT_PRFMSLC implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

SYSINSTR_128, bits [39:36]

SYSINSTR_128. Indicates support for System instructions that can take 128-bit inputs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYSINSTR_128	Meaning
0b0000	System instructions that can take 128-bit inputs are not supported.
0b0001	System instructions that can take 128-bit inputs are supported.

All other values are reserved.

FEAT_SYSINSTR128 implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

SYSREG_128, bits [35:32]

SYSREG_128. Indicates support for instructions to access 128-bit System Registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYSREG_128	Meaning
0b0000	Instructions to access 128-bit System Registers are not supported.
0b0001	Instructions to access 128-bit System Registers are supported.

All other values are reserved.

FEAT_SYSREG128 implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

CLRBHB, bits [31:28]

Indicates support for the CLRBHB instruction in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CLRBHB	Meaning
0b0000	CLRBHB instruction is not implemented.
0b0001	CLRBHB instruction is implemented.

All other values are reserved.

FEAT_CLRBHB implements the functionality identified by the value 0b0001.

From Armv8.9, the value 0b0000 is not permitted.

Access to this field is **RO**.

PAC_frac, bits [27:24]

Indicates which address bit is used to determine the size of the PAC field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PAC_frac	Meaning
0b0000	The address bit which is used to define the size of the PAC field is dependent on whether address tagging is used.
0b0001	The address bit which is used to define the size of the PAC field is fixed.

All other values are reserved.

FEAT_CONSTPACFIELD implements the functionality identified by the value 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

BC, bits [23:20]

Indicates support for the BC instruction in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BC	Meaning
0b0000	BC instruction is not implemented.
0b0001	BC instruction is implemented.

All other values are reserved.

FEAT_HBC implements the functionality identified by the value 0b0001.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is **RO**.

MOPS, bits [19:16]

Indicates support for the Memory Copy and Memory Set instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MOPS	Meaning
0b0000	The Memory Copy and Memory Set instructions are not implemented in AArch64 state.
0b0001	The Memory Copy and Memory Set instructions are implemented in AArch64 state with the following exception. If FEAT_MTE is implemented, then SETGP*, SETGM* and SETGE* instructions are also supported.

All other values are reserved.

FEAT_MOPS implements the functionality identified by the value 0b0001.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is **RO**.

APA3, bits [15:12]

Indicates whether the QARMA3 algorithm is implemented in the PE for address authentication in AArch64 state. This applies to all Pointer Authentication instructions other than the `PACGA` instruction.

The value of this field is an IMPLEMENTATION DEFINED choice of:

APA3	Meaning
`0b0000`	Address Authentication using the QARMA3 algorithm is not implemented.
`0b0001`	Address Authentication using the QARMA3 algorithm is implemented, FEAT_EPAC and FEAT_PAuth2 are not implemented.
`0b0010`	Address Authentication using the QARMA3 algorithm is implemented, FEAT_EPAC is implemented, and FEAT_PAuth2 is not implemented.
`0b0011`	Address Authentication using the QARMA3 algorithm is implemented, FEAT_EPAC is not implemented, and FEAT_PAuth2 is implemented.
`0b0100`	Address Authentication using the QARMA3 algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2 and FEAT_FPAC are implemented.
`0b0101`	Address Authentication using the QARMA3 algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, and FEAT_FPACCOMBINE are implemented.
`0b0110`	Address Authentication using the QARMA3 algorithm is implemented, including instructions that allow signing of LR using SP and PC as diversifiers, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, FEAT_FPACCOMBINE, and FEAT_PAuth_LR are implemented.

All other values are reserved.

FEAT_PAuth implements the functionality identified by the value `0b0001`.

FEAT_EPAC implements the functionality identified by the value `0b0010`.

FEAT_PAuth2 implements the functionality identified by the value `0b0011`.

FEAT_FPAC implements the functionality identified by the value `0b0100`.

FEAT_FPACCOMBINE implements the functionality identified by the value `0b0101`.

FEAT_PAuth_LR implements the functionality identified by the value `0b0110`.

When this field is nonzero, FEAT_PACQARMA3 is implemented.

In Armv8.3, the permitted values are `0b0000`, `0b0001`, `0b0010`, `0b0011`, `0b0100`, and `0b0101`.

From Armv8.6, the permitted values are `0b0000`, `0b0011`, `0b0100`, and `0b0101`.

From Armv9.5, the permitted values are `0b0000`, `0b0011`, `0b0100`, `0b0101`, and `0b0110`.

If the value of ID_AA64ISAR1_EL1.API is nonzero, or the value of ID_AA64ISAR1_EL1.APA is nonzero, this field must have the value `0b0000`.

Access to this field is **RO**.

GPA3, bits [11:8]

Indicates whether the QARMA3 algorithm is implemented in the PE for generic code authentication in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GPA3	Meaning
`0b0000`	Generic Authentication using the QARMA3 algorithm is not implemented.
`0b0001`	Generic Authentication using the QARMA3 algorithm is implemented. This includes the `PACGA` instruction.

All other values are reserved.

FEAT_PACQARMA3 implements the functionality identified by the value 0b0001.

When this field is nonzero, FEAT_PACQARMA3 is implemented.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPI is nonzero, or the value of ID_AA64ISAR1_EL1.GPA is nonzero, this field must have the value 0b0000.

Access to this field is **RO**.

RPRES, bits [7:4]

Indicates support for 12 bits of mantissa in single-precision reciprocal and reciprocal square root instructions in AArch64 state, when FPCR.AH is 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RPRES	Meaning
0b0000	Single-precision reciprocal and reciprocal square root estimates give 8 bits of mantissa, when FPCR.AH is 1.
0b0001	Single-precision reciprocal and reciprocal square root estimates give 12 bits of mantissa, when FPCR.AH is 1.

All other values are reserved.

FEAT_RPRES implements the functionality identified by the value 0b0001.

When FPCR.AH is 0, the floating-point reciprocal estimate and reciprocal square root estimate instructions give 8 bits of mantissa.

Access to this field is **RO**.

WFxT, bits [3:0]

Indicates support for the WFET and WFIT instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WFxT	Meaning
0b0000	WFET and WFIT are not supported.
0b0010	WFET and WFIT are supported, and the register number is reported in the ESR_ELx on exceptions.

All other values are reserved.

FEAT_WFxT implements the functionality identified by the value 0b0010.

From Armv8.7, the only permitted value is 0b0010.

Access to this field is **RO**.

Accessing ID_AA64ISAR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64ISAR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0110	0b010

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64ISAR2_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_AA64ISAR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64ISAR2_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64ISAR3_EL1, AArch64 Instruction Set Attribute Register 3

The ID_AA64ISAR3_EL1 characteristics are:

Purpose

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64ISAR3_EL1 are UNDEFINED.

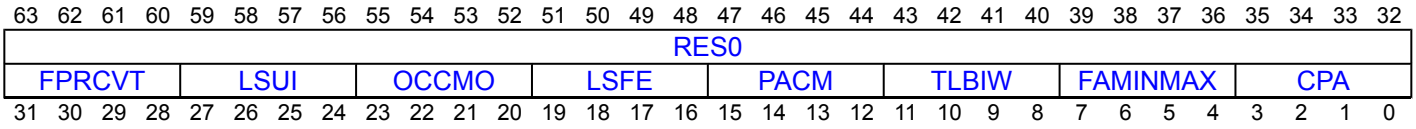
Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64ISAR3_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

FPRCVT, bits [31:28]

Indicates support for the following conversions between floating-point and integer instructions with only scalar SIMD&FP register operands and results, and with different input and output register sizes:

- FCVTAS (scalar SIMD&FP), FCVTAU (scalar SIMD&FP).
- FCVTMS (scalar SIMD&FP), FCVTMU (scalar SIMD&FP).
- FCVTNS (scalar SIMD&FP), FCVTNU (scalar SIMD&FP).
- FCVTPS (scalar SIMD&FP), FCVTPU (scalar SIMD&FP).
- FCVTZS (scalar SIMD&FP), FCVTZU (scalar SIMD&FP).
- SCVTF (scalar SIMD&FP), UCVTF (scalar SIMD&FP).

If FEAT_SME2p1 is implemented, FEAT_FPRCVT indicates support for the following conversions between floating-point and integer instructions with only scalar SIMD&FP register operands and results, and with the same input and output register sizes when the PE is in Streaming SVE mode:

- FCVTAS (vector), FCVTAU (vector).
- FCVTMS (vector), FCVTMU (vector).
- FCVTNS (vector), FCVTNU (vector).
- FCVTPS (vector), FCVTPU (vector).
- FCVTZS (vector), FCVTZU (vector).

- SCVTF (vector), UCVTF (vector).

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPRCVT	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

FEAT_FPRCVT implements the functionality identified by the value 0b0001.

From Armv9.6, the value 0b0000 is not permitted.

Access to this field is **RO**.

LSUI, bits [27:24]

Support for the following load and store unprivileged instructions:

- LDTXR, STTXR.
- CAST, CASAT, CASALT, CASLT.
- CASP, CASPA, CASPAL, CASPL.
- LDTP, STTP.
- LDTP (SIMD&FP), STTP (SIMD&FP).
- LDTNP, STTNP.
- LDTNP (SIMD&FP), STTNP (SIMD&FP).

The value of this field is an IMPLEMENTATION DEFINED choice of:

LSUI	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

FEAT_LSUI implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

OCCMO, bits [23:20]

Indicates support for the following cache maintenance operation to the Outer cache level instructions:

- DC CIVAOC.
- DC CVAOC.

If FEAT_MTE is implemented:

- DC CIGDVAOC.
- DC CGDVAOC.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OCCMO	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

FEAT_OCCMO implements the functionality identified by the value 0b0001.

From Armv9.6, the value 0b0000 is not permitted.

Access to this field is **RO**.

LSFE, bits [19:16]

Indicates support for the following A64 Base atomic floating-point in-memory instructions:

- Floating-point atomic add in memory: LDFADD, LDFADDA, LDFADDAL, and LDFADDL.
- BFloat16 atomic add in memory: LDBFADD, LDBFADDA, LDBFADDAL, and LDBFADDL.

- Floating-point atomic maximum in memory: LDFMAX, LDFMAXA, LDFMAXAL, and LDFMAXL.
- BFloat16 atomic maximum in memory: LDBFMAX, LDBFMAXA, LDBFMAXAL, and LDBFMAXL.
- Floating-point atomic maximum number in memory: LDFMAXNM, LDFMAXNMA, LDFMAXNMAL, and LDFMAXNML.
- BFloat16 atomic maximum number in memory: LDBFMAXNM, LDBFMAXNMA, LDBFMAXNMAL, and LDBFMAXNML.
- Floating-point atomic minimum in memory: LDFMIN, LDFMINA, LDFMINAL, and LDFMINL.
- BFloat16 atomic minimum in memory: LDBFMIN, LDBFMINA, LDBFMINAL, and LDBFMINL.
- Floating-point atomic minimum number in memory: LDFMINNM, LDFMINNMA, LDFMINNMAL, and LDFMINNML.
- BFloat16 atomic minimum number in memory: LDBFMINNM, LDBFMINNMA, LDBFMINNMAL, and LDBFMINNML.
- Floating-point atomic add in memory, without return: STFADD and STFADDL.
- BFloat16 atomic add in memory, without return: STBFADD and STBFADDL.
- Floating-point atomic maximum in memory, without return: STFMAX and STFMAXL.
- BFloat16 atomic maximum in memory, without return: STBFMAX and STBFMAXL.
- Floating-point atomic maximum number in memory, without return: STFMAXNM and STFMAXNML.
- BFloat16 atomic maximum number in memory, without return: STBFMAXNM and STBFMAXNML.
- Floating-point atomic minimum in memory, without return: STFMIN and STFMINL.
- BFloat16 atomic minimum in memory, without return: STBFMIN and STBFMINL.
- Floating-point atomic minimum number in memory, without return: STFMINNM and STFMINNML.
- BFloat16 atomic minimum number in memory, without return: STBFMINNM and STBFMINNML.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LSFE	Meaning
0b0000	Atomic floating-point in-memory instructions are not implemented.
0b0001	The specified Atomic floating-point in-memory instructions are implemented.

FEAT_LSFE implements the functionality identified by the value 0b0001

Access to this field is **RO**.

PACM, bits [15:12]

Indicates the implementation of PSTATE.PACM.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PACM	Meaning
0b0000	PSTATE.PACM is not implemented.
0b0001	Trivial implementation of PSTATE.PACM.
0b0010	Full implementation of PSTATE.PACM.

All other values are reserved.

FEAT_PAuth_LR implements the functionality identified by the values 0b0001 and 0b0010.

If FEAT_PAuth_LR is implemented, the value 0b0000 is not permitted.

If ID_AA64ISAR1_EL1.API is 0b0000, the value 0b0001 is not permitted.

If one of FEAT_PACQARMA3 or FEAT_PACQARMA5 are implemented, the value 0b0001 is not permitted.

Access to this field is **RO**.

TLBIW, bits [11:8]

Support for TLBI VMALL for Dirty state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TLBIW	Meaning
0b0000	TLBI VMALL for Dirty state is not supported.
0b0001	TLBI VMALL for Dirty state is supported.

All other values are reserved.

FEAT_TLBIW implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

FAMINMAX, bits [7:4]

Indicates support for the following Advanced SIMD, SVE2, and SME2 instructions that compute maximum and minimum absolute value:

- When Advanced SIMD is implemented, the Advanced SIMD instructions `FAMAX` and `FAMIN`.
- When FEAT_SVE2 or FEAT_SME2 is implemented, the SVE2 instructions `FAMAX` and `FAMIN`.
- When FEAT_SME2 is implemented, the SME2 instructions `FAMAX` (multiple and single vectors), `FAMIN` (multiple and single vectors), `FAMAX` (multiple vectors), and `FAMIN` (multiple vectors).

The value of this field is an IMPLEMENTATION DEFINED choice of:

FAMINMAX	Meaning
`0b0000`	The specified instructions are not implemented.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_FAMINMAX implements the functionality identified by the value `0b0001`.

Access to this field is **RO**.

CPA, bits [3:0]

Indicates support for Checked Pointer Arithmetic instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CPA	Meaning
`0b0000`	Checked Pointer Arithmetic instructions are not implemented.
`0b0001`	Checked Pointer Arithmetic instructions are implemented.
`0b0010`	Checked Pointer Arithmetic instructions are implemented, and Checked Pointer Arithmetic can be enabled.

All other values are reserved.

FEAT_CPA implements the functionality identified by the value `0b0001`.

FEAT_CPA2 implements the functionality identified by the value `0b0010`.

From Armv9.5, the value `0b0000` is not permitted.

Access to this field is **RO**.

Accessing ID_AA64ISAR3_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64ISAR3_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0110	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64ISAR3_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_AA64ISAR3_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR3_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR3_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64ISAR3_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0

The ID_AA64MMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

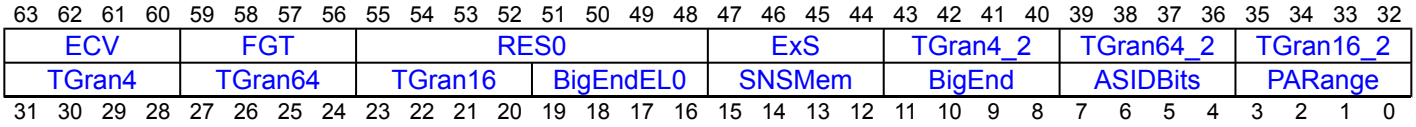
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64MMFR0_EL1 are UNDEFINED.

Attributes

ID_AA64MMFR0_EL1 is a 64-bit register.

Field descriptions



ECV, bits [63:60]

Indicates presence of Enhanced Counter Virtualization.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ECV	Meaning
0b0000	Enhanced Counter Virtualization is not implemented.
0b0001	Enhanced Counter Virtualization is implemented. Supports CNTHCTL_EL2.{EL1TVT, EL1TVCT, EL1NVPCT, EL1NVVCT, EVNTIS}, CNTKCTL_EL1.EVNTIS, CNTPCTSS_EL0 counter views, and CNTVCTSS_EL0 counter views. Extends the PMSCR_EL1.PCT, PMSCR_EL2.PCT, TRFCR_EL1.TS, and TRFCR_EL2.TS fields.
0b0010	As 0b0001, and the CNTPOFF_EL2 register and the CNTHCTL_EL2.ECV and SCR_EL3.ECVEn fields are implemented.

All other values are reserved.

FEAT_ECV implements the functionality identified by the value 0b0001.

FEAT_ECV_POFF implements the functionality identified by the value 0b0010.

From Armv8.6, the value 0b0000 is not permitted.

Access to this field is **RO**.

FGT, bits [59:56]

Indicates presence of the Fine-Grained Trap controls.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FGT	Meaning
0b0000	Fine-grained trap controls are not implemented.
0b0001	Fine-grained trap controls are implemented. Supports: • If EL2 is implemented, the HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGRTR_EL2, HFGITR_EL2 and HFGWTR_EL2 registers, and their associated traps. • If EL2 is implemented, MDCR_EL2.TDCC. • If EL3 is implemented, MDCR_EL3.TDCC. • If both EL2 and EL3 are implemented, SCR_EL3.FGTEn.
0b0010	As 0b0001, and also includes support for: • If EL2 is implemented, the HDFGRTR2_EL2, HDFGWTR2_EL2, HFGITR2_EL2, HFGRTR2_EL2, and HFGWTR2_EL2 registers, and their associated traps. • If both EL2 and EL3 are implemented, SCR_EL3.FGTEn2.

All other values are reserved.

FEAT_FGT implements the functionality identified by the value 0b0001.

FEAT_FGT2 implements the functionality identified by the value 0b0010.

From Armv8.6, the value 0b0000 is not permitted.

From Armv8.9, the value 0b0001 is not permitted.

Access to this field is **RO**.

Bits [55:48]

Reserved, RES0.

ExS, bits [47:44]

Indicates support for disabling context synchronizing exception entry and exit.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ExS	Meaning
0b0000	All exception entries and exits are context synchronization events.
0b0001	Non-context synchronizing exception entry and exit are supported.

All other values are reserved.

FEAT_ExS implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

TGran4_2, bits [43:40]

Indicates support for 4KB memory granule size at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TGran4_2	Meaning	Applies when
0b0000	Support for 4KB granule at stage 2 is identified in the ID_AA64MMFR0_EL1.TGran4 field.	
0b0001	4KB granule not supported at stage 2.	
0b0010	4KB granule supported at stage 2.	
0b0011	4KB granule at stage 2 supports 52-bit input addresses and can describe 52-bit output addresses.	When FEAT_LPA2 is implemented

All other values are reserved.

If EL2 is not implemented, this field is 0b0000. Otherwise, the value 0b0000 is deprecated.

Note

This field does not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64MMFR0_EL1 stage 2 granule sizes for more information.

Access to this field is **RO**.

TGran64_2, bits [39:36]

Indicates support for 64KB memory granule size at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TGran64_2	Meaning
0b0000	Support for 64KB granule at stage 2 is identified in the ID_AA64MMFR0_EL1.TGran64 field.
0b0001	64KB granule not supported at stage 2.
0b0010	64KB granule supported at stage 2.

All other values are reserved.

If EL2 is not implemented, this field is 0b0000. Otherwise, the value 0b0000 is deprecated.

Note

This field does not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64MMFR0_EL1 stage 2 granule sizes for more information.

Access to this field is **RO**.

TGran16_2, bits [35:32]

Indicates support for 16KB memory granule size at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TGran16_2	Meaning	Applies when
0b0000	Support for 16KB granule at stage 2 is identified in the ID_AA64MMFR0_EL1.TGran16 field.	
0b0001	16KB granule not supported at stage 2.	
0b0010	16KB granule supported at stage 2.	
0b0011	16KB granule at stage 2 supports 52-bit input addresses and can describe 52-bit output addresses.	When FEAT_LPA2 is implemented

All other values are reserved.

If EL2 is not implemented, this field is 0b0000. Otherwise, the value 0b0000 is deprecated.

Note

This field does not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64MMFR0_EL1 stage 2 granule sizes for more information.

Access to this field is **RO**.

TGran4, bits [31:28]

Indicates support for 4KB memory translation granule size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TGran4	Meaning	Applies when
0b0000	4KB granule supported.	
0b0001	4KB granule supports 52-bit input addresses and can describe 52-bit output addresses.	When FEAT_LPA2 is implemented
0b1111	4KB granule not supported.	

All other values are reserved.

Access to this field is **RO**.

TGran64, bits [27:24]

Indicates support for 64KB memory translation granule size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TGran64	Meaning
0b0000	64KB granule supported.
0b1111	64KB granule not supported.

All other values are reserved.

Access to this field is **RO**.

TGran16, bits [23:20]

Indicates support for 16KB memory translation granule size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TGran16	Meaning	Applies when
0b0000	16KB granule not supported.	
0b0001	16KB granule supported.	
0b0010	16KB granule supports 52-bit input addresses and can describe 52-bit output addresses.	When FEAT_LPA2 is implemented

All other values are reserved.

Access to this field is **RO**.

BigEndEL0, bits [19:16]

Indicates support for mixed-endian at EL0 only.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BigEndEL0	Meaning
0b0000	No mixed-endian support at EL0. The SCTLR_EL1.E0E bit has a fixed value.
0b0001	Mixed-endian support at EL0. The SCTLR_EL1.E0E bit can be configured.

FEAT_MixedEndEL0 implements the functionality identified by the value 0b0001.

All other values are reserved.

This field is invalid and is RES0 if ID_AA64MMFR0_EL1.BigEnd is not 0b0000.

Access to this field is **RO**.

SNSMem, bits [15:12]

Indicates support for a distinction between Secure and Non-secure Memory.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SNSMem	Meaning
0b0000	Does not support a distinction between Secure and Non-secure Memory.
0b0001	Does support a distinction between Secure and Non-secure Memory.

Note

If EL3 is implemented, the value 0b0000 is not permitted.

All other values are reserved.

Access to this field is **RO**.

BigEnd, bits [11:8]

Indicates support for mixed-endian configuration.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BigEnd	Meaning
0b0000	No mixed-endian support. The SCTLR_ELx.EE bits have a fixed value. See the BigEndEL0 field, bits[19:16], for whether EL0 supports mixed-endian.
0b0001	Mixed-endian support. The SCTLR_ELx.EE and SCTLR_EL1.E0E bits can be configured.

FEAT_MixedEnd implements the functionality identified by the value 0b0001.

If this field is 0b0001, FEAT_MixedEndEL0 is also implemented.

All other values are reserved.

Access to this field is **RO**.

ASIDBits, bits [7:4]

Number of ASID bits.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ASIDBits	Meaning
0b0000	8 bits.
0b0010	16 bits.

All other values are reserved.

Access to this field is **RO**.

PARange, bits [3:0]

Physical Address range supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PARange	Meaning	Applies when
0b0000	32 bits, 4GB.	
0b0001	36 bits, 64GB.	
0b0010	40 bits, 1TB.	
0b0011	42 bits, 4TB.	
0b0100	44 bits, 16TB.	
0b0101	48 bits, 256TB.	
0b0110	52 bits, 4PB.	When FEAT_LPA is implemented
0b0111	56 bits, 64PB.	When FEAT_D128 is implemented

All other values are reserved.

Access to this field is **RO**.

Accessing ID_AA64MMFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64MMFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64MMFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1

The ID_AA64MMFR1_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

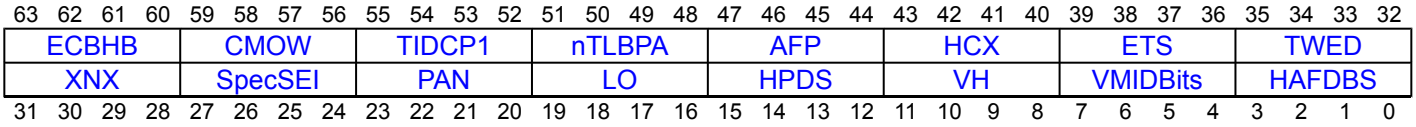
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64MMFR1_EL1 are UNDEFINED.

Attributes

ID_AA64MMFR1_EL1 is a 64-bit register.

Field descriptions



ECBHB, bits [63:60]

Indicates support for restrictions on branch history speculation around exceptions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ECBHB	Meaning
0b0000	The implementation does not disclose whether restrictions are imposed on branch history speculation around exceptions.
0b0001	The implementation imposes restrictions on branch history speculation around exceptions.

All other values are reserved.

FEAT_ECBHB implements the functionality identified by the value 0b0001.

From Armv8.9, the value 0b0000 is not permitted.

Access to this field is **RO**.

CMOW, bits [59:56]

Indicates support for cache maintenance instruction permission.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CMOW	Meaning
0b0000	SCTLR_EL1.CMOW, SCTLR_EL2.CMOW, and HCRX_EL2.CMOW bits are not implemented.
0b0001	SCTLR_EL1.CMOW is implemented. If EL2 is implemented, SCTLR_EL2.CMOW and HCRX_EL2.CMOW bits are implemented.

All other values are reserved.

FEAT_CMOW implements the functionality identified by the value 0b0001.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is **RO**.

TIDCP1, bits [55:52]

Indicates whether SCTLR_EL1.TIDCP and SCTLR_EL2.TIDCP are implemented in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TIDCP1	Meaning
0b0000	SCTLR_EL1.TIDCP and SCTLR_EL2.TIDCP bits are not implemented and are RES0.
0b0001	SCTLR_EL1.TIDCP bit is implemented. If EL2 is implemented, SCTLR_EL2.TIDCP bit is implemented.

All other values are reserved.

FEAT_TIDCP1 implements the functionality identified by the value 0b0001.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is **RO**.

nTLBPA, bits [51:48]

Indicates support for intermediate caching of translation table walks.

The value of this field is an IMPLEMENTATION DEFINED choice of:

nTLBPA	Meaning
0b0000	The intermediate caching of translation table walks might include non-coherent physical translation caches.
0b0001	The intermediate caching of translation table walks does not include non-coherent physical translation caches.

Non-coherent physical translation caches are non-coherent caches of previous valid translation table entries since the last completed relevant TLBI applicable to the PE, where either:

- The caching is indexed by the physical address of the location holding the translation table entry.
- The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry.

All other values are reserved.

FEAT_nTLBPA implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

AFP, bits [47:44]

Indicates support for FPCR.{AH, FIZ, NEP}.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AFP	Meaning
0b0000	The FPCR.{AH, FIZ, NEP} fields are not supported.
0b0001	The FPCR.{AH, FIZ, NEP} fields are supported.

All other values are reserved.

FEAT_AFP implements the functionality identified by the value 0b0001.

From Armv8.7, if Advanced SIMD and floating-point is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

HCX, bits [43:40]

Indicates support for HCRX_EL2 and its associated EL3 trap.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HCX	Meaning
0b0000	HCRX_EL2 and its associated EL3 trap are not supported.
0b0001	HCRX_EL2 and its associated EL3 trap are supported.

All other values are reserved.

FEAT_HCX implements the functionality identified by the value 0b0001.

From Armv8.7, if EL2 is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

ETS, bits [39:36]

Indicates support for Enhanced Translation Synchronization.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ETS	Meaning
0b0000	Enhanced Translation Synchronization is not supported.
0b0001	Enhanced Translation Synchronization is not supported.
0b0010	FEAT_ETS2 is implemented.
0b0011	FEAT_ETS3 is implemented.

All other values are reserved.

FEAT_ETS2 implements the functionality identified by the value 0b0010.

FEAT_ETS3 implements the functionality identified by the value 0b0011.

From Armv8.8, the values 0b0000 and 0b0001 are not permitted.

From Armv9.5, the value 0b0010 is not permitted.

Access to this field is **RO**.

TWED, bits [35:32]

Indicates support for the configurable delayed trapping of WFE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TWED	Meaning
0b0000	Configurable delayed trapping of WFE is not supported.
0b0001	Configurable delayed trapping of WFE is supported.

All other values are reserved.

FEAT_TWED implements the functionality identified by the value 0b0001.

From Armv8.6, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

XNX, bits [31:28]

Indicates support for execute-never control distinction by Exception level at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

XNX	Meaning
0b0000	Distinction between EL0 and EL1 execute-never control at stage 2 not supported.
0b0001	Distinction between EL0 and EL1 execute-never control at stage 2 supported.

All other values are reserved.

FEAT_XNX implements the functionality identified by the value 0b0001.

From Armv8.2, the value 0b0000 is not permitted.

Access to this field is **RO**.

SpecSEI, bits [27:24]

When FEAT_RAS is implemented:

Describes whether the PE can generate SError exceptions from speculative reads of memory, including speculative instruction fetches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SpecSEI	Meaning
0b0000	The PE never generates an SError exception due to an External abort on a speculative read.
0b0001	The PE might generate an SError exception due to an External abort on a speculative read.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

PAN, bits [23:20]

Privileged Access Never. Indicates support for the PAN bit in PSTATE, SPSR_EL1, SPSR_EL2, SPSR_EL3, and DSPSR_EL0.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PAN	Meaning
0b0000	PAN not supported.
0b0001	PAN supported.
0b0010	PAN supported and AT S1E1RP and AT S1E1WP instructions supported.
0b0011	PAN supported, AT S1E1RP and AT S1E1WP instructions supported, and SCTLR_EL1.EPAN and SCTLR_EL2.EPAN bits supported.

All other values are reserved.

FEAT_PAN implements the functionality identified by the value 0b0001.

FEAT_PAN2 implements the functionality added by the value 0b0010.

FEAT_PAN3 implements the functionality added by the value 0b0011.

From Armv8.1, the value 0b0000 is not permitted.

From Armv8.2, the value 0b0001 is not permitted.

From Armv8.7, the value 0b0010 is not permitted.

Access to this field is **RO**.

LO, bits [19:16]

LORegions. Indicates support for LORegions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LO	Meaning
0b0000	LORegions not supported.
0b0001	LORegions supported.

All other values are reserved.

FEAT_LOR implements the functionality identified by the value 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

HPDS, bits [15:12]

Hierarchical Permission Disables. Indicates support for disabling hierarchical controls in translation tables.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HPDS	Meaning
0b0000	Disabling of hierarchical controls not supported.
0b0001	Disabling of hierarchical controls supported with the TCR_EL1.{HPD1, HPD0}, TCR_EL2.HPD or TCR_EL2.{HPD1, HPD0}, and TCR_EL3.HPD bits.
0b0010	As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the Translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use.

All other values are reserved.

FEAT_HPDS implements the functionality identified by the value 0b0001.

FEAT_HPDS2 implements the functionality identified by the value 0b0010.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

VH, bits [11:8]

Virtualization Host Extensions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VH	Meaning
0b0000	Virtualization Host Extensions not supported.
0b0001	Virtualization Host Extensions supported.

All other values are reserved.

FEAT_VHE implements the functionality identified by the value 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

VMIDBits, bits [7:4]

Number of VMID bits.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMIDBits	Meaning
0b0000	8 bits
0b0010	16 bits

All other values are reserved.

FEAT_VMID16 implements the functionality identified by the value 0b0010.

From Armv8.1, the permitted values are 0b0000 and 0b0010.

Access to this field is **RO**.

HAFDBS, bits [3:0]

Hardware updates to Access flag and Dirty state in translation tables.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAFDBS	Meaning
0b0000	Hardware update of the Access flag and dirty state are not supported.
0b0001	Support for hardware update of the Access flag for Block and Page descriptors.
0b0010	As 0b0001, and adds support for hardware update of dirty state.
0b0011	As 0b0010, and adds support for hardware update of the Access flag for Table descriptors.
0b0100	As 0b0011, and adds support for hardware tracking of Dirty state Structure.

All other values are reserved.

FEAT_HAFDBS implements the functionality identified by the values 0b0001 and 0b0010.

FEAT_HAFT implements the functionality identified by the value 0b0011.

FEAT_HDBSS implements the functionality identified by the value 0b0100.

Access to this field is **RO**.

Accessing ID_AA64MMFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64MMFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0111	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64MMFR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2

The ID_AA64MMFR2_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64MMFR2_EL1 are UNDEFINED.

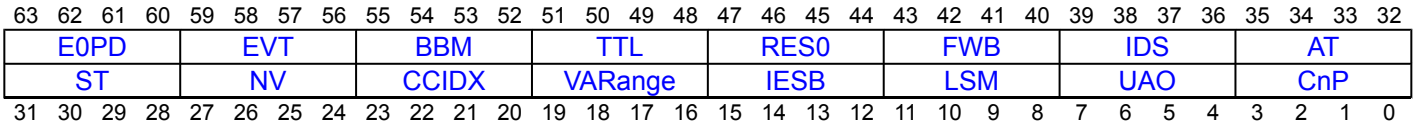
Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64MMFR2_EL1 is a 64-bit register.

Field descriptions



E0PD, bits [63:60]

Indicates support for the E0PD mechanism.

The value of this field is an IMPLEMENTATION DEFINED choice of:

E0PD	Meaning
0b0000	E0PDx mechanism is not implemented.
0b0001	E0PDx mechanism is implemented.

All other values are reserved.

FEAT_E0PD implements the functionality identified by the value 0b0001.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

If FEAT_E0PD is implemented, FEAT_CSV3 must be implemented.

Access to this field is **RO**.

EVT, bits [59:56]

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the HCR_EL2.{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EVT	Meaning
0b0000	HCR_EL2.{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps are not supported.
0b0001	HCR_EL2.{TOCU, TICAB, TID4} traps are supported. HCR_EL2.{TTLBOS, TTLBIS} traps are not supported.
0b0010	HCR_EL2.{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps are supported.

All other values are reserved.

FEAT_EVT implements the functionality identified by the values 0b0001 and 0b0010.

If EL2 is not implemented, the only permitted value is 0b0000.

From Armv8.5, if EL2 is implemented, the value 0b0001 is not permitted.

Access to this field is **RO**.

BBM, bits [55:52]

Allows identification of the requirements of the hardware to have break-before-make sequences when changing block or table size for a translation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BBM	Meaning
0b0000	Break-before-make sequence must be used.
0b0001	Level 1 support for changing block size is supported.
0b0010	Level 2 support for changing block size is supported.

All other values are reserved.

FEAT_BBML1 implements the functionality identified by the value 0b0001.

FEAT_BBML2 implements the functionality identified by the value 0b0010.

Access to this field is **RO**.

TTL, bits [51:48]

Indicates support for TTL field in address operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TTL	Meaning
0b0000	TLB maintenance instructions by address have bits[47:44] as RES0.
0b0001	TLB maintenance instructions by address have bits[47:44] holding the TTL field.

All other values are reserved.

FEAT_TTL implements the functionality identified by the value 0b0001.

This field affects TLBI IPAS2E1, TLBI IPAS2E1IS, TLBI IPAS2E1OS, TLBI IPAS2LE1, TLBI IPAS2LE1IS, TLBI IPAS2LE1OS, TLBI VAAE1, TLBI VAAE1IS, TLBI VAAE1OS, TLBI VAALE1, TLBI VAALE1IS, TLBI VAALE1OS, TLBI VAE1, TLBI VAE1IS, TLBI VAE1OS, TLBI VAE2, TLBI VAE2IS, TLBI VAE2OS, TLBI VAE3, TLBI VAE3IS, TLBI VAE3OS,TLBI VALE1, TLBI VALE1IS, TLBI VALE1OS, TLBI VALE2, TLBI VALE2IS, TLBI VALE2OS, TLBI VALE3, TLBI VALE3IS, TLBI VALE3OS.

From Armv8.4, the value 0b0000 is not permitted.

Access to this field is **RO**.

Bits [47:44]

Reserved, RES0.

FWB, bits [43:40]

Indicates support for HCR_EL2.FWB.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FWB	Meaning
0b0000	HCR_EL2.FWB bit is not supported.
0b0001	HCR_EL2.FWB is supported.

All other values reserved.

FEAT_S2FWB implements the functionality identified by the value 0b0001.

From Armv8.4, the value 0b0000 is not permitted.

Access to this field is **RO**.

IDS, bits [39:36]

Indicates the EC syndrome value in ESR_ELx.EC that is reported if an exception is generated by a read access to the feature ID space.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IDS	Meaning
0b0000	An exception which is generated by a read access to the feature ID space, other than a trap caused by HCR_EL2.TIDx, SCTLR_EL1.UCT, or SCTLR_EL2.UCT is reported by ESR_ELx.EC == 0x0.
0b0001	All exceptions generated by an AArch64 read access to the feature ID space are reported by ESR_ELx.EC == 0x18.
0b0010	As 0b0001 and introduces support for trapping ID register accesses to EL3.

All other values are reserved.

The Feature ID space is defined as the System register space in AArch64 with op0==3, op1=={0, 1, 3}, CRn==0, CRm=={0-7}, op2=={0-7}.

FEAT_IDST implements the functionality identified by the value 0b0001.

FEAT_IDTE3 implements the functionality identified by 0b0010.

From Armv8.4, the value 0b0000 is not permitted.

From Armv9.6, if EL3 is implemented, the value 0b0001 is not permitted.

Access to this field is **RO**.

AT, bits [35:32]

Identifies support for unaligned single-copy atomicity and atomic functions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AT	Meaning
0b0000	Unaligned single-copy atomicity and atomic functions are not supported.
0b0001	Unaligned single-copy atomicity and atomic functions with a 16-byte address range aligned to 16-bytes are supported.

All other values are reserved.

FEAT_LSE2 implements the functionality identified by the value 0b0001.

From Armv8.4, the value 0b0000 is not permitted.

Access to this field is **RO**.

ST, bits [31:28]

Identifies support for small translation tables.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ST	Meaning
0b0000	The maximum value of the TCR_ELx.{T0SZ,T1SZ} and VTCR_EL2.T0SZ fields is 39.
0b0001	The maximum value of the TCR_ELx.{T0SZ,T1SZ} and VTCR_EL2.T0SZ fields is 48 for 4KB and 16KB granules, and 47 for 64KB granules.

All other values are reserved.

FEAT_TTST implements the functionality identified by the value 0b0001.

When FEAT_SEL2 is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

NV, bits [27:24]

Nested Virtualization. If EL2 is implemented, indicates support for the use of nested virtualization.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NV	Meaning
0b0000	If ID_AA64MMFR4_EL1.NV_frac != 0b0000, support for nested virtualization is described in ID_AA64MMFR4_EL1.NV_frac. Otherwise, nested virtualization is not supported.
0b0001	The HCR_EL2.{AT, NV1, NV} bits are implemented.
0b0010	The VNCR_EL2 register and the HCR_EL2.{NV2, AT, NV1, NV} bits are implemented.

All other values are reserved.

If EL2 is not implemented, the only permitted value is 0b0000.

FEAT_NV implements the functionality identified by the value 0b0001.

FEAT_NV2 implements the functionality identified by the value 0b0010.

In Armv8.3, if EL2 is implemented, the permitted values are 0b0000 and 0b0001.

From Armv8.4, if EL2 is implemented, the permitted values are 0b0000, 0b0001, and 0b0010.

Access to this field is **RO**.

CCIDX, bits [23:20]

Support for the use of revised CCSIDR_EL1 register format.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCIDX	Meaning
0b0000	32-bit format implemented for all levels of the CCSIDR_EL1.
0b0001	64-bit format implemented for all levels of the CCSIDR_EL1.

All other values are reserved.

FEAT_CCIDX implements the functionality identified by the value 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

VARange, bits [19:16]

Indicates support for a larger virtual address.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VARange	Meaning	Applies when
0b0000	VMSAv8-64 supports 48-bit VAs.	
0b0001	VMSAv8-64 supports 52-bit VAs when using the 64KB translation granule. The size for other translation granules is not defined by this field.	
0b0010	VMSAv9-128 supports 56-bit VAs.	When FEAT_D128 is implemented

All other values are reserved.

FEAT_LVA implements the functionality identified by the value 0b0001.

FEAT_LVA3 implements the functionality identified by the value 0b0010.

Access to this field is **RO**.

IESB, bits [15:12]

Indicates support for the IESB bit in the SCTLR_ELx registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IESB	Meaning
0b0000	IESB bit in the SCTLR_ELx registers is not supported.
0b0001	IESB bit in the SCTLR_ELx registers is supported.

All other values are reserved.

FEAT_IESB implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

LSM, bits [11:8]

Indicates support for LSMAOE and nTLSMD bits in SCTLR_EL1 and SCTLR_EL2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LSM	Meaning
0b0000	LSMAOE and nTLSMD bits not supported.
0b0001	LSMAOE and nTLSMD bits supported.

All other values are reserved.

FEAT_LSMAOC implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

UAO, bits [7:4]

User Access Override.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UAO	Meaning
0b0000	UAO not supported.
0b0001	UAO supported.

All other values are reserved.

FEAT_UAO implements the functionality identified by the value `0b0001`.

From Armv8.2, the value `0b0000` is not permitted.

Access to this field is **RO**.

CnP, bits [3:0]

Indicates support for Common not Private translations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CnP	Meaning
`0b0000`	Common not Private translations not supported.
`0b0001`	Common not Private translations supported.

All other values are reserved.

FEAT_TTCNP implements the functionality identified by the value `0b0001`.

From Armv8.2, the value `0b0000` is not permitted.

Access to this field is **RO**.

Accessing ID_AA64MMFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64MMFR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0111	0b010

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64MMFR2_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_AA64MMFR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64MMFR2_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64MMFR3_EL1, AArch64 Memory Model Feature Register 3

The ID_AA64MMFR3_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64MMFR3_EL1 are UNDEFINED.

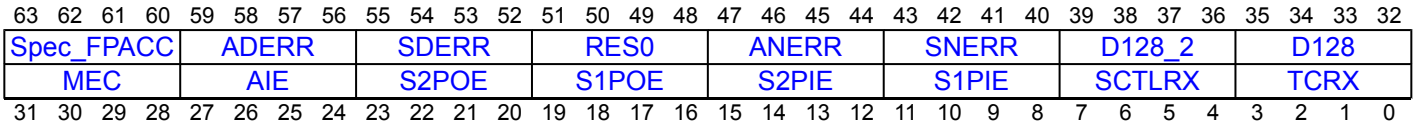
Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64MMFR3_EL1 is a 64-bit register.

Field descriptions



Spec_FPACC, bits [63:60]

When FEAT_FPACCOMBINE is implemented:

Speculative behavior in the event of a PAC authentication failure in an implementation that includes FEAT_FPACCOMBINE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Spec_FPACC	Meaning
0b0000	The implementation does not disclose whether the speculative use of pointers processed by a PAC Authentication is materially different in terms of the impact on cached microarchitectural state between passing and failing of the PAC Authentication.
0b0001	The speculative use of pointers processed by a PAC Authentication is not materially different in terms of the impact on cached microarchitectural state between passing and failing of the PAC Authentication.

All other values are reserved.

For the purpose of this definition, cached microarchitecture state is the state of caching agents such as instruction caches, data caches and TLBs which can be altered as a result of speculation caused by a mispredicted execution, but is not restored to the state prior to the speculation when the misprediction is corrected.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

ADERR, bits [59:56]

Asynchronous Device error exceptions. With ID_AA64MMFR3_EL1.SDERR, describes the PE behavior for External aborts signaled on Device memory loads.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ADERR	Meaning
0b0000	If FEAT_RASv2 is not implemented and ID_AA64MMFR3_EL1.SDERR is 0b0000, then the behavior is not described. Otherwise, the behavior is described by ID_AA64MMFR3_EL1.SDERR.
0b0001	All External aborts on Device memory loads are handled asynchronously.
0b0010	FEAT_ADERR is implemented. SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR are implemented. If FEAT_ANERR is also implemented, then software should ensure that all the following apply: • SCTLR2_ELx.EnADERR is set to the value of SCTLR2_ELx.EnANERR. • HCRX_EL2.EnSDERR is set to the value of HCRX_EL2.EnSNERR.
0b0011	FEAT_ADERR is implemented. SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR are implemented. If FEAT_ANERR is also implemented, then SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR operate independently of SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR.

All other values are reserved.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

When FEAT_RASv2 is implemented and ID_AA64MMFR3_EL1.SDERR is 0b0000, the value of this field is 0b0001.

When ID_AA64MMFR3_EL1.SDERR is 0b0001, the value of this field is 0b0000.

When ID_AA64MMFR3_EL1.SDERR is 0b0010, the value of this field is 0b0010.

When ID_AA64MMFR3_EL1.SDERR is 0b0011, the value of this field is 0b0011.

FEAT_ADERR implements the functionality described by the value 0b0010.

Access to this field is **RO**.

SDERR, bits [55:52]

Synchronous Device error exceptions. With ID_AA64MMFR3_EL1.ADERR, describes the PE behavior for External aborts signaled on Device memory loads.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SDERR	Meaning
0b0000	If FEAT_RASv2 is not implemented and ID_AA64MMFR3_EL1.ADERR is 0b0000, then the behavior is not described. Otherwise, the behavior is described by ID_AA64MMFR3_EL1.ADERR.
0b0001	All External aborts on Device memory loads are handled synchronously.
0b0010	FEAT_ADERR is implemented. SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR are implemented. If FEAT_ANERR is also implemented, then software should ensure that all the following apply: • SCTLR2_ELx.EnADERR is set to the value of SCTLR2_ELx.EnANERR. • HCRX_EL2.EnSDERR is set to the value of HCRX_EL2.EnSNERR.
0b0011	FEAT_ADERR is implemented. SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR are implemented. If FEAT_ANERR is also implemented, then SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR operate independently of SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR.

All other values are reserved.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

When FEAT_RASv2 is implemented and ID_AA64MMFR3_EL1.ADERR is 0b0000, the value of this field is 0b0001.

When ID_AA64MMFR3_EL1.ADERR is 0b0001, the value of this field is 0b0000.

When ID_AA64MMFR3_EL1.ADERR is 0b0010, the value of this field is 0b0010.

When ID_AA64MMFR3_EL1.ADERR is 0b0011, the value of this field is 0b0011.

FEAT_ADERR implements the functionality described by the value 0b0010.

Access to this field is **RO**.

Bits [51:48]

Reserved, RES0.

ANERR, bits [47:44]

Asynchronous Normal error exceptions. With ID_AA64MMFR3_EL1.SNERR, describes the PE behavior for External aborts signaled on Normal memory loads.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ANERR	Meaning
0b0000	If FEAT_RASv2 is not implemented and ID_AA64MMFR3_EL1.SNERR is 0b0000, then the behavior is not described. Otherwise, the behavior is described by ID_AA64MMFR3_EL1.SNERR.
0b0001	All External aborts on Normal memory loads are handled asynchronously.
0b0010	FEAT_ANERR is implemented. SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR are implemented. If FEAT_ADERR is also implemented, then software should ensure that all the following apply: • SCTLR2_ELx.EnANERR is set to the value of SCTLR2_ELx.EnADERR. • HCRX_EL2.EnSNERR is set to the value of HCRX_EL2.EnSDERR.
0b0011	FEAT_ANERR is implemented. SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR are implemented. If FEAT_ADERR is also implemented, then SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR operate independently of SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR.

All other values are reserved.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

When FEAT_RASv2 is implemented and ID_AA64MMFR3_EL1.SNERR is 0b0000, the value of this field is 0b0001.

When ID_AA64MMFR3_EL1.SNERR is 0b0001, the value of this field is 0b0000.

When ID_AA64MMFR3_EL1.SNERR is 0b0010, the value of this field is 0b0010.

When ID_AA64MMFR3_EL1.SNERR is 0b0011, the value of this field is 0b0011.

FEAT_ANERR implements the functionality described by the value 0b0010.

Access to this field is **RO**.

SNERR, bits [43:40]

Synchronous Normal error exceptions. With ID_AA64MMFR3_EL1.ANERR, describes the PE behavior for External aborts signaled on Normal memory loads.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SNERR	Meaning
0b0000	If FEAT_RASv2 is not implemented and ID_AA64MMFR3_EL1.ANERR is 0b0000, then the behavior is not described. Otherwise, the behavior is described by ID_AA64MMFR3_EL1.ANERR.
0b0001	All External aborts on Normal memory loads are handled synchronously.
0b0010	FEAT_ANERR is implemented. SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR are implemented. If FEAT_ADERR is also implemented, then software should ensure that all the following apply: • SCTLR2_ELx.EnANERR is set to the value of SCTLR2_ELx.EnADERR. • HCRX_EL2.EnSNERR is set to the value of HCRX_EL2.EnSDERR.
0b0011	FEAT_ANERR is implemented. SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR are implemented. If FEAT_ADERR is also implemented, then SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR operate independently of SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR.

All other values are reserved.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

When FEAT_RASv2 is implemented and ID_AA64MMFR3_EL1.ANERR is 0b0000, the value of this field is 0b0001.

When ID_AA64MMFR3_EL1.ANERR is 0b0001, the value of this field is 0b0000.

When ID_AA64MMFR3_EL1.ANERR is 0b0010, the value of this field is 0b0010.

When ID_AA64MMFR3_EL1.ANERR is 0b0011, the value of this field is 0b0011.

FEAT_ANERR implements the functionality described by the value 0b0010.

Access to this field is **RO**.

D128_2, bits [39:36]

128-bit translation table descriptor at stage 2. Indicates support for 128-bit translation table descriptor at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

D128_2	Meaning
0b0000	128-bit translation table descriptor Extension at stage 2 is not supported.
0b0001	128-bit translation table descriptor Extension at stage 2 is supported.

All other values are reserved.

Access to this field is **RO**.

D128, bits [35:32]

128-bit translation table descriptor. Indicates support for 128-bit translation table descriptor.

The value of this field is an IMPLEMENTATION DEFINED choice of:

D128	Meaning
0b0000	128-bit translation table descriptor Extension is not supported.
0b0001	128-bit translation table descriptor Extension is supported.

All other values are reserved.

Access to this field is **RO**.

MEC, bits [31:28]

Indicates support for Memory Encryption Contexts.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MEC	Meaning
0b0000	Memory Encryption Contexts is not supported.
0b0001	Memory Encryption Contexts is supported, with multiple contexts in the Realm physical address space.

All other values are reserved.

FEAT_MEC implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

AIE, bits [27:24]

Attribute Indexing. Indicates support for the Attribute Index Enhancement.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AIE	Meaning
0b0000	The Attribute Index Enhancement is not supported.
0b0001	The Attribute Index Enhancement at stage 1 is supported.

All other values are reserved.

FEAT_AIE implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

S2POE, bits [23:20]

Stage 2 Permission Overlay. Indicates support for Permission Overlay at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

S2POE	Meaning
0b0000	Permission Overlay at stage 2 is not supported.
0b0001	Permission Overlay at stage 2 is supported.

All other values are reserved.

FEAT_S2POE implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

S1POE, bits [19:16]

Stage 1 Permission Overlay. Indicates support for Permission Overlay at stage 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

S1POE	Meaning
0b0000	Permission Overlay at stage 1 is not supported.
0b0001	Permission Overlay at stage 1 is supported.

All other values are reserved.

FEAT_S1POE implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

S2PIE, bits [15:12]

Stage 2 Permission Indirection. Indicates support for Permission Indirection at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

S2PIE	Meaning
0b0000	Permission Indirection at stage 2 is not supported.
0b0001	Permission Indirection at stage 2 is supported.

All other values are reserved.

FEAT_S2PIE implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

S1PIE, bits [11:8]

Stage 1 Permission Indirection. Indicates support for Permission Indirection at stage 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

S1PIE	Meaning
0b0000	Permission Indirection at stage 1 is not supported.
0b0001	Permission Indirection at stage 1 is supported.

All other values are reserved.

FEAT_S1PIE implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

SCTLRX, bits [7:4]

SCTLR Extension. Indicates support for extension of SCTLR_ELx.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SCTLRX	Meaning
0b0000	SCTLR2_EL1, SCTLR2_EL2, SCTLR2_EL3 registers, and their associated trap controls are not implemented.
0b0001	SCTLR2_EL1, SCTLR2_EL2, SCTLR2_EL3 resisters, and their associated trap controls are implemented.

All other values are reserved.

From Armv8.9, the value 0b0000 is not permitted.

FEAT_SCTLR2 implements the functionality described by the value 0b0001.

Access to this field is **RO**.

TCRX, bits [3:0]

TCR Extension. Indicates support for extension of TCR_ELx.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TCRX	Meaning
0b0000	TCR2_EL1, TCR2_EL2, and their associated trap controls are not implemented.
0b0001	TCR2_EL1, TCR2_EL2, and their associated trap controls are implemented.

All other values are reserved.

From Armv8.9, the value 0b0000 is not permitted.

FEAT_TCR2 implements the functionality described by the value 0b0001.

Access to this field is **RO**.

Accessing ID_AA64MMFR3_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64MMFR3_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0111	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64MMFR3_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_AA64MMFR3_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR3_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR3_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64MMFR3_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64MMFR4_EL1, AArch64 Memory Model Feature Register 4

The ID_AA64MMFR4_EL1 characteristics are:

Purpose

Provides additional information about implemented memory model and memory management support in AArch64.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64MMFR4_EL1 are UNDEFINED.

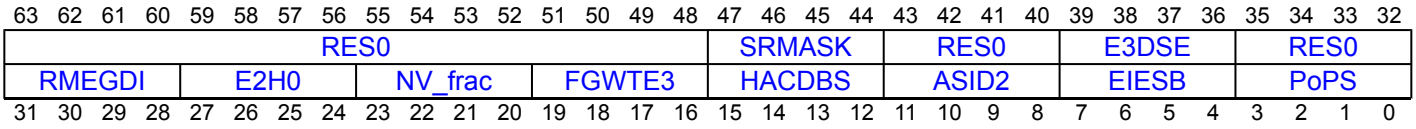
Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64MMFR4_EL1 is a 64-bit register.

Field descriptions



Bits [63:48]

Reserved, RES0.

SRMASK, bits [47:44]

Indicates support for bitwise write masks for the following registers:

- ACTLR_EL1.
- CPACR_EL1.
- SCTLR_EL1.
- SCTLR2_EL1.
- TCR_EL1.
- TCR2_EL1.
- ACTLR_EL2.
- CPTR_EL2.
- SCTLR_EL2.
- SCTLR2_EL2.
- TCR_EL2.
- TCR2_EL2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SRMASK	Meaning
0b0000	Bitwise write masks for the specified registers are not supported.
0b0001	Bitwise write masks for the specified registers are supported.

FEAT_SRMASK implements the functionality identified by the value 0b0001.

From Armv9.6, the value 0b0000 is not permitted.

Access to this field is **RO**.

Bits [43:40]

Reserved, RES0.

E3DSE, bits [39:36]

Delegated SError exceptions from EL3. Describes support for delegated SError injection from EL3.

The value of this field is an IMPLEMENTATION DEFINED choice of:

E3DSE	Meaning
0b0000	FEAT_E3DSE is not implemented.
0b0001	FEAT_E3DSE is implemented. The following are implemented: • Register fields SCR_EL3.DSE and SCR_EL3.EnDSE. • Registers VSESR_EL3 and VDISR_EL3.

All other values are reserved.

FEAT_E3DSE implements the functionality described by the value 0b0001.

Access to this field is **RO**.

Bits [35:32]

Reserved, RES0.

RMEGDI, bits [31:28]

RME Granular Data Isolation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RMEGDI	Meaning
0b0000	The Granular Data isolation GPI encodings are reserved.
0b0001	The Granular Data Isolation GPI encodings can be configured to be either reserved or No Access from this PE.

All other values are reserved.

FEAT_RME_GDI implements the functionality described by the value 0b0001.

Access to this field is **RO**.

E2H0, bits [27:24]

Indicates support for programming HCR_EL2.E2H.

The value of this field is an IMPLEMENTATION DEFINED choice of:

E2H0	Meaning
0b0000	FEAT_E2H0 is implemented.
0b1110	FEAT_E2H0 is not implemented. HCR_EL2.NV1 is RES0.
0b1111	FEAT_E2H0 is not implemented.

All other values are reserved.

If FEAT_NV is not implemented, then the value 0b1110 is not permitted.

If FEAT_E2H0 is implemented and FEAT_VHE is not implemented, then HCR_EL2.E2H is RES0.

If FEAT_E2H0 is implemented and FEAT_VHE is implemented, then HCR_EL2.E2H can be programmed to 0 or 1.

If FEAT_E2H0 is not implemented then:

- FEAT_VHE is implemented.
- HCR_EL2.E2H is RES1 and behaves as though it is 1.

Access to this field is **RO**.

NV_frac, bits [23:20]

Indicates support for a subset of FEAT_NV and FEAT_NV2 behaviors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NV_frac	Meaning
0b0000	Support for FEAT_NV and FEAT_NV2 is described in ID_AA64MMFR2_EL1.NV.
0b0001	FEAT_NV and FEAT_NV2 are implemented, but all of the following apply: • ID_AA64MMFR2_EL1.NV is 0b0000. • Programming HCR_EL2.{NV, NV2} to {1, 0} behaves as {1, 1}.
0b0010	As 0b0001 and adds stateful bits in specified _EL1 registers for software usage under nested virtualization.

FEAT_NV2p1 implements the functionality indicated by the value 0b0010.

Access to this field is **RO**.

FGWTE3, bits [19:16]

Indicates support for Fine Grained Write Trap EL3 feature.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FGWTE3	Meaning
0b0000	Fine Grained Write Trap EL3 is not supported.
0b0001	Fine Grained Write Trap EL3 is supported.

All other values are reserved.

FEAT_FGWTE3 implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

HACDBS, bits [15:12]

Support for Hardware accelerator for cleaning Dirty state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HACDBS	Meaning
0b0000	Hardware accelerator for cleaning Dirty state is not supported.
0b0001	Hardware accelerator for cleaning Dirty state is supported.

All other values are reserved.

FEAT_HACDBS implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

ASID2, bits [11:8]

Indicates support for concurrent use of two ASIDs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ASID2	Meaning
0b0000	FEAT_ASID2 is not implemented.
0b0001	FEAT_ASID2 is implemented.

All other values are reserved.

From Armv9.5, the value 0b0000 is not permitted.

Access to this field is **RO**.

EIESB, bits [7:4]

When FEAT_IESB is implemented:

Early Implicit Error Synchronization event. Indicates whether the implicit Error synchronization event inserted on taking an exception to ELx when SCTLR_ELx.IESB is 1 is inserted before or after the exception is taken.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EIESB	Meaning
0b1111	FEAT_IESB is implemented. When SError exceptions are routed to EL3, and either FEAT_DoubleFault is not implemented or the Effective value of SCR_EL3.NMEA is 1, an implicit Error synchronization event might be inserted after an exception is taken to EL3.
0b0000	FEAT_IESB is not implemented, or behavior is not described.
0b0001	FEAT_IESB is implemented. When SError exceptions are routed to EL3, and either FEAT_DoubleFault is not implemented or the Effective value of SCR_EL3.NMEA is 1, an implicit Error synchronization event is inserted before an exception taken to EL3.
0b0010	As 0b0001, and also: • When SError exceptions are routed to EL1, and either FEAT_DoubleFault2 is not implemented or the Effective value of SCTLR2_EL1.NMEA is 1, an implicit Error synchronization event is inserted before an exception taken to EL1. • When SError exceptions are routed to EL2, and either FEAT_DoubleFault2 is not implemented or the Effective value of SCTLR2_EL2.NMEA is 1, an implicit Error synchronization event is inserted before an exception taken to EL2.

All other values are reserved.

This field describes the PE behavior on taking an exception to ELx when SCTLR_ELx.IESB is 1. This field does not describe the behavior when SCTLR_ELx.IESB is 0.

The behavior described by this field only applies for the conditions described above. For example, if ID_AA64MMFR4_EL1.EIESB reads as 0b0001, then it does not describe the behavior when SError exceptions are not routed to EL3, or when FEAT_DoubleFault is implemented and the Effective value of SCR_EL3.NMEA is 0.

Inserting the event before the exception is taken means that if the Error synchronization event causes an SError exception to become pending, and SError exceptions are not masked and not disabled, then the SError exception is taken in place of the original exception.

When FEAT_IESB is not implemented, the only permitted value of this field is 0b0000.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

PoPS, bits [3:0]

Support for the clean and invalidate to the Point of Physical Storage instructions:

- DC CIVAPS.
- If FEAT_MTE2 is implemented, DC CIGDVAPS.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PoPS	Meaning
0b0000	The System instructions to clean and invalidate to the Point of Physical Storage are not implemented.
0b0001	The specified System instructions to clean and invalidate to the Point of Physical Storage are implemented.

FEAT_PoPS implements the functionality described by the value 0b0001.

All other values are reserved.

Access to this field is **RO**.

Accessing ID_AA64MMFR4_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64MMFR4_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0111	0b100

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64MMFR4_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_AA64MMFR4_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR4_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR4_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64MMFR4_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0

The ID_AA64PFR0_EL1 characteristics are:

Purpose

Provides additional information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

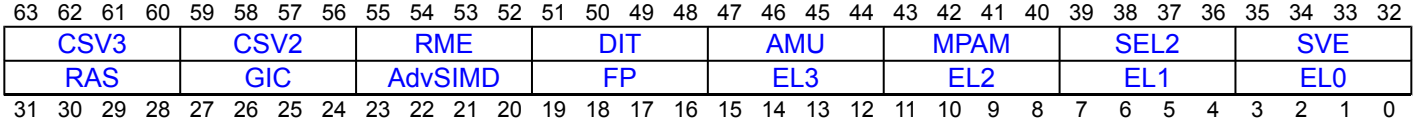
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64PFR0_EL1 are UNDEFINED.

The external register EDPFR gives information from this register.

Attributes

ID_AA64PFR0_EL1 is a 64-bit register.

Field descriptions



CSV3, bits [63:60]

Speculative use of faulting data.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV3	Meaning
0b0000	This PE does not disclose whether data loaded or read from a register under speculation where the data load or register read would not be permitted architecturally, can be used by instructions newer than the load or register read in a manner that allows the value of the inaccessible data to be recovered by code architecturally executed.
0b0001	Data loaded or read from a register under speculation where the data load or register read would not be permitted architecturally, cannot be used by instructions newer than the load or register read in a manner that allows the value of the inaccessible data to be recovered by code architecturally executed.

All other values are reserved.

FEAT_CSV3 implements the functionality identified by the value 0b0001.

If FEAT_E0PD is implemented, FEAT_CSV3 must be implemented.

From Armv8.5, the value 0b0000 is not permitted.

Access to this field is **RO**.

CSV2, bits [59:56]

Speculative use of out of context prediction resources.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV2	Meaning
0b0000	The implementation does not disclose whether FEAT_CSV2 is implemented.
0b0001	FEAT_CSV2 is implemented, but FEAT_CSV2_2 and FEAT_CSV2_3 are not implemented. ID_AA64PFR1_EL1.CSV2_frac determines whether either or both of FEAT_CSV2_1p1 or FEAT_CSV2_1p2 are implemented.
0b0010	FEAT_CSV2_2 is implemented, but FEAT_CSV2_3 is not implemented.
0b0011	FEAT_CSV2_3 is implemented.

All other values are reserved.

FEAT_CSV2 implements the functionality identified by the value 0b0001.

FEAT_CSV2_2 implements the functionality identified by the value 0b0010.

FEAT_CSV2_3 implements the functionality identified by the feature 0b0011.

From Armv8.5, the value 0b0000 is not permitted.

Access to this field is **RO**.

RME, bits [55:52]

Realm Management Extension (RME).

The value of this field is an IMPLEMENTATION DEFINED choice of:

RME	Meaning
0b0000	Realm Management Extension not implemented.
0b0001	RMEv1 is implemented.
0b0010	As 0b0001, and adds support for the GPC2 Extension.
0b0011	As 0b0010, and adds support for the GPC3 Extension.

All other values are reserved.

FEAT_RME implements the functionality identified by the value 0b0001.

FEAT_RME_GPC2 implements the functionality identified by the value 0b0010.

FEAT_RME_GPC3 implements the functionality identified by the value 0b0011.

Access to this field is **RO**.

DIT, bits [51:48]

Data Independent Timing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DIT	Meaning
0b0000	AArch64 does not guarantee constant execution time of any instructions.
0b0001	AArch64 provides the PSTATE.DIT mechanism to guarantee constant execution time of certain instructions.

All other values are reserved.

FEAT_DIT implements the functionality identified by the value 0b0001.

From Armv8.4, the value 0b0000 is not permitted.

Access to this field is **RO**.

AMU, bits [47:44]

Indicates support for Activity Monitors Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AMU	Meaning
0b0000	Activity Monitors Extension is not implemented.
0b0001	FEAT_AMUv1 is implemented.
0b0010	FEAT_AMUv1p1 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters.

All other values are reserved.

FEAT_AMUv1 implements the functionality identified by the value 0b0001.

FEAT_AMUv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0000.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.6, the permitted values are 0b0000, 0b0001, and 0b0010.

Access to this field is **RO**.

MPAM, bits [43:40]

Indicates the major version number of support for the MPAM Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MPAM	Meaning
0b0000	The major version number of the MPAM extension is 0.
0b0001	The major version number of the MPAM extension is 1.

All other values are reserved.

When combined with the minor version number from ID_AA64PFR1_EL1.MPAM_frac, the "major.minor" version is:

MPAM Extension version	MPAM	MPAM_frac
Not implemented.	0b0000	0b0000
v0.1 is implemented.	0b0000	0b0001
v1.0 is implemented.	0b0001	0b0000
v1.1 is implemented.	0b0001	0b0001

For more information, see 'The Memory Partitioning and Monitoring (MPAM) Extension'.

Access to this field is **RO**.

SEL2, bits [39:36]

Secure EL2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEL2	Meaning
0b0000	Secure EL2 is not implemented.
0b0001	Secure EL2 is implemented.

All other values are reserved.

FEAT_SEL2 implements the functionality identified by the value 0b0001.

From Armv8.4, if Secure state and EL2 are implemented, the value 0b0000 is not permitted.

From Armv8.4, if Secure state is not implemented, or if EL2 is not implemented, the only permitted value is 0b0000.

Access to this field is **RO**.

SVE, bits [35:32]

Scalable Vector Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SVE	Meaning
0b0000	SVE architectural state and programmers' model are not implemented.
0b0001	SVE architectural state and programmers' model are implemented.

All other values are reserved.

FEAT_SVE implements the functionality identified by the value 0b0001.

If implemented, refer to ID_AA64ZFR0_EL1 for information about which SVE instructions are available.

Access to this field is **RO**.

RAS, bits [31:28]

RAS Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RAS	Meaning
0b0000	No RAS Extension.
0b0001	Support for the Reliability, Availability, and Serviceability Extension is implemented. The ESB instruction and the Error synchronization event are supported.
0b0010	As 0b0001, and adds support for: • If EL3 is implemented, FEAT_DoubleFault. • Additional ERXMISC<m>_EL1 System registers. • Additional System registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1, and the SCR_EL3.FIEN and HCR_EL2.FIEN trap controls, to support the optional RAS Common Fault Injection Model Extension. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp and RAS Common Fault Injection Model Extensions.
0b0011	As 0b0010 and adds support for: • ERXGSR_EL1, to support System RAS agents. • Additional fine-grained EL2 traps for additional error record System registers. • The SCR_EL3.TWERR write control for error record System registers. Error records accessed through System registers conform to RAS System Architecture v2.

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0001.

FEAT_RASv1p1 and FEAT_DoubleFault implement the functionality identified by the value 0b0010.

FEAT_RASv2 implements the functionality identified by the value 0b0011.

In Armv8.0 and Armv8.1, the permitted values are 0b0000 and 0b0001.

From Armv8.2, the value 0b0000 is not permitted.

From Armv8.4, if FEAT_DoubleFault is implemented or ERRIDR_EL1.NUM is nonzero, the value 0b0001 is not permitted.

Note

When the value of this field is 0b0001, ID_AA64PFR1_EL1.RAS_frac indicates whether FEAT_RASv1p1 is implemented.

From Armv8.9, if FEAT_DoubleFault is implemented or ERRIDR_EL1.NUM is nonzero, the value 0b0010 is not permitted.

Access to this field is **RO**.

GIC, bits [27:24]

System register GIC CPU interface.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GIC	Meaning
0b0000	GIC CPU interface system registers not implemented.
0b0001	System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.
0b0011	System register interface to version 4.1 of the GIC CPU interface is supported.

All other values are reserved.

Access to this field is **RO**.

AdvSIMD, bits [23:20]

Advanced SIMD.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AdvSIMD	Meaning
0b0000	Advanced SIMD is implemented, including support for the following SISD and SIMD operations: • Integer byte, halfword, word and doubleword element operations. • Single-precision and double-precision floating-point arithmetic. • Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.
0b0001	As for 0b0000, and also includes support for half-precision floating-point arithmetic.
0b1111	Advanced SIMD is not implemented.

All other values are reserved.

This field must have the same value as the FP field.

The permitted values are:

- 0b0000 in an implementation with Advanced SIMD support that does not include the FEAT_FP16 extension.
- 0b0001 in an implementation with Advanced SIMD support that includes the FEAT_FP16 extension.
- 0b1111 in an implementation without Advanced SIMD support.

Access to this field is **RO**.

FP, bits [19:16]

Floating-point.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FP	Meaning
0b0000	Floating-point is implemented, and includes support for: • Single-precision and double-precision floating-point types. • Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.
0b0001	As for 0b0000, and also includes support for half-precision floating-point arithmetic.
0b1111	Floating-point is not implemented.

All other values are reserved.

This field must have the same value as the AdvSIMD field.

The permitted values are:

- `0b0000` in an implementation with floating-point support that does not include the FEAT_FP16 extension.
- `0b0001` in an implementation with floating-point support that includes the FEAT_FP16 extension.
- `0b1111` in an implementation without floating-point support.

Access to this field is **RO**.

EL3, bits [15:12]

EL3 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL3	Meaning
`0b0000`	EL3 is not implemented.
`0b0001`	EL3 can be executed in AArch64 state only.
`0b0010`	EL3 can be executed in either AArch64 or AArch32 state.

All other values are reserved.

The value `0b0010` is not permitted in Armv9-A implementations.

Access to this field is **RO**.

EL2, bits [11:8]

EL2 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL2	Meaning
`0b0000`	EL2 is not implemented.
`0b0001`	EL2 can be executed in AArch64 state only.
`0b0010`	EL2 can be executed in either AArch64 or AArch32 state.

All other values are reserved.

The value `0b0010` is not permitted in Armv9-A implementations.

Access to this field is **RO**.

EL1, bits [7:4]

EL1 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL1	Meaning
`0b0001`	EL1 can be executed in AArch64 state only.
`0b0010`	EL1 can be executed in either AArch64 or AArch32 state.

All other values are reserved.

The value `0b0010` is not permitted in Armv9-A implementations.

Access to this field is **RO**.

EL0, bits [3:0]

EL0 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL0	Meaning
0b0001	EL0 can be executed in AArch64 state only.
0b0010	EL0 can be executed in either AArch64 or AArch32 state.

All other values are reserved.

Access to this field is **RO**.

Accessing ID_AA64PFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64PFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64PFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1

The ID_AA64PFR1_EL1 characteristics are:

Purpose

Provides additional information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

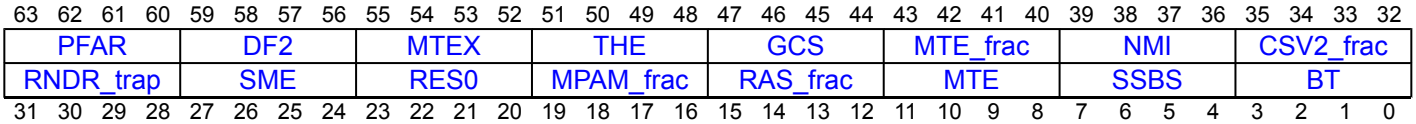
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64PFR1_EL1 are UNDEFINED.

Attributes

ID_AA64PFR1_EL1 is a 64-bit register.

Field descriptions



PFAR, bits [63:60]

Support for physical fault address registers, FEAT_PFAR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PFAR	Meaning
0b0000	FEAT_PFAR is not implemented.
0b0001	FEAT_PFAR is implemented. Includes support for the PFAR_ELx and, if EL3 is implemented, MFAR_EL3 registers.

All other values are reserved.

FEAT_PFAR implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

DF2, bits [59:56]

Support for error exception routing extensions, FEAT_DoubleFault2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DF2	Meaning
0b0000	FEAT_DoubleFault2 is not implemented. **Note** This does not mean that FEAT_DoubleFault, as identified by ID_AA64PFR0_EL1.RAS >= 0b0010, is not implemented.
0b0001	FEAT_DoubleFault2 is implemented. As ID_AA64PFR0_EL1.RAS == 0b0010, and also includes support for routing error exceptions: • Traps for masked error exceptions, HCRX_EL2.TMEA and SCR_EL3.TMEA. • Additional controls for masking SError exceptions, SCTLR2_EL1.NMEA, and SCTLR2_EL2.NMEA. • Additional controls for taking external aborts to the SError exception vector, SCTLR2_EL1.EASE and SCTLR2_EL2.EASE.

All other values are reserved.

FEAT_DoubleFault2 implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

MTEX, bits [55:52]

Support for additional MTE tag checking modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTEX	Meaning
0b0000	Canonical Tag checking and Memory tagging with Address tagging disabled are not supported.
0b0001	The following additional tag checking modes for MTE are supported: • Canonical Tag checking. • Memory tagging with Address tagging disabled.

All other values are reserved.

This field is valid only if ID_AA64PFR1_EL1.MTE >= 0b0010.

FEAT_MTE_NO_ADDRESS_TAGS and FEAT_MTE_CANONICAL_TAGS implement the functionality identified by the value 0b0001.

If FEAT_MTE2 is not implemented, the value 0b0001 is not permitted.

From Armv8.9, if FEAT_MTE2 is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

THE, bits [51:48]

Support for Translation Hardening Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

THE	Meaning
0b0000	Translation Hardening Extension is not implemented.
0b0001	The RCW and RCWS instructions, their associated registers and traps are supported. If EL2 is implemented, the AssuredOnly check, TopLevel check, and their associated controls are implemented. If EL2 and FEAT_GCS are implemented, VTCR_EL2.GCSH is implemented.

All other values are reserved.

FEAT_THE implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

GCS, bits [47:44]

Support for Guarded Control Stack.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GCS	Meaning
0b0000	Guarded Control Stack is not implemented.
0b0001	Guarded Control Stack is implemented.

All other values are reserved.

FEAT_GCS implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

MTE_frac, bits [43:40]

Support for Asynchronous reporting of a Tag Check Fault.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTE_frac	Meaning
0b0000	Asynchronous reporting of a Tag Check Fault is supported.
0b1111	Asynchronous reporting of a Tag Check Fault is not supported.

All other values are reserved.

This field is valid only if ID_AA64PFR1_EL1.MTE >= 0b0010.

FEAT_MTE_ASYNC implements the functionality identified by the value 0b0000.

If FEAT_MTE_ASYM_FAULT is implemented this field must be 0b0000.

Access to this field is **RO**.

NMI, bits [39:36]

Non-maskable Interrupt. Indicates support for Non-maskable interrupts.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NMI	Meaning
0b0000	SCTLR_ELx.{SPINTMASK, NMI} and PSTATE.ALLINT with its associated instructions are not supported.
0b0001	SCTLR_ELx.{SPINTMASK, NMI} and PSTATE.ALLINT with its associated instructions are supported.

All other values are reserved.

FEAT_NMI implements the functionality identified by the value 0b0001.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is **RO**.

CSV2_frac, bits [35:32]

CSV2 fractional field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV2_frac	Meaning
0b0000	Either ID_AA64PFR0_EL1.CSV2 is not 0b0001, or the implementation does not disclose whether FEAT_CSV2_1p1 is implemented. FEAT_CSV2_1p2 is not implemented.
0b0001	FEAT_CSV2_1p1 is implemented, but FEAT_CSV2_1p2 is not implemented.
0b0010	FEAT_CSV2_1p2 is implemented.

All other values are reserved.

FEAT_CSV2_1p1 implements the functionality identified by the value 0b0001.

FEAT_CSV2_1p2 implements the functionality identified by the value 0b0010.

From Armv8.0, the permitted values are 0b0000, 0b0001, and 0b0010.

The values 0b0001 and 0b0010 are permitted only when ID_AA64PFR0_EL1.CSV2 is 0b0001.

Access to this field is **RO**.

RNDR_trap, bits [31:28]

Random Number trap to EL3 field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RNDR_trap	Meaning
0b0000	Trapping of RNDR and RNDRRS to EL3 is not supported.
0b0001	Trapping of RNDR and RNDRRS to EL3 is supported. SCR_EL3.TRNDR is present.

All other values are reserved.

FEAT_RNG_TRAP implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

SME, bits [27:24]

Scalable Matrix Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SME	Meaning
0b0000	SME architectural state and programmers' model are not implemented.
0b0001	SME architectural state and programmers' model are implemented.
0b0010	As 0b0001, plus the SME2 ZT0 register.

All other values are reserved.

FEAT_SME implements the functionality identified by the value 0b0001.

FEAT_SME2 implements the functionality identified by the value 0b0010.

From Armv9.2, the permitted values are 0b0000, 0b0001, and 0b0010.

If implemented, refer to ID_AA64SMFR0_EL1 and ID_AA64ZFR0_EL1 for information about which SME and SVE instructions are available.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

MPAM_frac, bits [19:16]

Indicates the minor version number of support for the MPAM Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MPAM_frac	Meaning
0b0000	The minor version number of the MPAM extension is 0.
0b0001	The minor version number of the MPAM extension is 1.

All other values are reserved.

When combined with the major version number from ID_AA64PFR0_EL1.MPAM, The combined "major.minor" version is:

MPAM Extension version	MPAM	MPAM_frac
Not implemented.	0b0000	0b0000
v0.1 is implemented.	0b0000	0b0001
v1.0 is implemented.	0b0001	0b0000
v1.1 is implemented.	0b0001	0b0001

For more information, see 'The Memory Partitioning and Monitoring (MPAM) Extension'.

Access to this field is **RO**.

RAS_frac, bits [15:12]

RAS Extension fractional field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RAS_frac	Meaning
0b0000	If ID_AA64PFR0_EL1.RAS == 0b0001, support for the Reliability, Availability, and Serviceability Extension is implemented.
0b0001	If ID_AA64PFR0_EL1.RAS == 0b0001, as 0b0000 and adds support for: • Additional ERXMISC<m>_EL1 System registers. • Additional System registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1, and the SCR_EL3.FIEN and HCR_EL2.FIEN trap controls, to support the optional RAS Common Fault Injection Model Extension. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS, and support for the optional RAS Timestamp and RAS Common Fault Injection Model Extensions.

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0000.

FEAT_RASv1p1 implements the functionality identified by the value 0b0001.

This field is valid only if ID_AA64PFR0_EL1.RAS == 0b0001.

Access to this field is **RO**.

MTE, bits [11:8]

Support for the Memory Tagging Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTE	Meaning
0b0000	Memory Tagging Extension is not implemented.
0b0001	Instruction-only Memory Tagging Extension is implemented.
0b0010	As 0b0001, and adds: • Support for in-memory Allocation tags. • Support for synchronous tag checking. • Optional support for Asynchronous reporting of a Tag Check Fault, identified as FEAT_MTE_ASYNC. Support for FEAT_MTE_ASYNC is indicated by ID_AA64PFR1_EL1.MTE_frac.
0b0011	As 0b0010, except that support for FEAT_MTE_ASYNC is mandatory, and adds support for Asymmetric Tag Check Fault handling, identified as FEAT_MTE_ASYM_FAULT.

All other values are reserved.

FEAT_MTE implements the functionality identified by the value 0b0001.

FEAT_MTE2 implements the functionality identified by the value 0b0010.

FEAT_MTE3 implements the functionality identified by the value 0b0011.

From Armv8.7, when the value of this field is >= 0b0010, ID_AA64PFR2_EL1.MTEPERM indicates support for FEAT_MTE_PERM.

From Armv8.7, when the value of this field is >= 0b0010, the following fields indicate support for FEAT_MTE4:

- ID_AA64PFR1_EL1.MTEX
- ID_AA64PFR2_EL1.MTEFAR
- ID_AA64PFR2_EL1.MTESTOREONLY

Access to this field is **RO**.

SSBS, bits [7:4]

Speculative Store Bypassing controls in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SSBS	Meaning
0b0000	AArch64 provides no mechanism to control the use of Speculative Store Bypassing.
0b0001	AArch64 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe.
0b0010	As 0b0001, and adds the MSR and MRS instructions to directly read and write the PSTATE.SSBS field.

All other values are reserved.

FEAT_SSBS implements the functionality identified by the value 0b0001.

FEAT_SSBS2 implements the functionality identified by the value 0b0010.

Access to this field is **RO**.

BT, bits [3:0]

Branch Target Identification mechanism support in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BT	Meaning
0b0000	The Branch Target Identification mechanism is not implemented.
0b0001	The Branch Target Identification mechanism is implemented.

All other values are reserved.

FEAT_BTI implements the functionality identified by the value 0b0001.

From Armv8.5, the value `0b0000` is not permitted.

Access to this field is **RO**.

Accessing ID_AA64PFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64PFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64PFR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64PFR2_EL1, AArch64 Processor Feature Register 2

The ID_AA64PFR2_EL1 characteristics are:

Purpose

Provides additional information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64PFR2_EL1 are UNDEFINED.

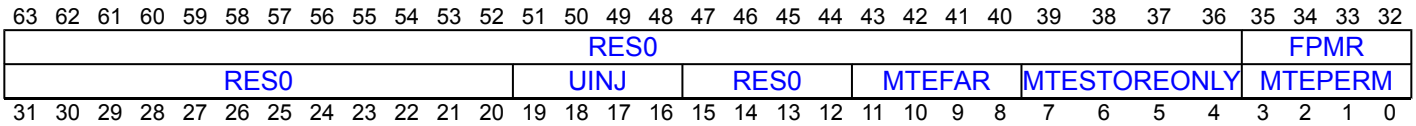
Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64PFR2_EL1 is a 64-bit register.

Field descriptions



Bits [63:36]

Reserved, RES0.

FPMR, bits [35:32]

Indicates support for FPMR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPMR	Meaning
0b0000	FPMR is not implemented.
0b0001	FPMR is implemented.

All other values are reserved.

FEAT_FPMR implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

Bits [31:20]

Reserved, RES0.

UINJ, bits [19:16]

Support for software injection of Undefined Instruction exceptions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UINJ	Meaning
0b0000	Software injection of Undefined Instruction exceptions is not implemented.
0b0001	Software injection of Undefined Instruction exceptions is implemented.

FEAT_UINJ implements the functionality identified by the value 0b0001.

From Armv9.6, the value 0b0000 is not permitted.

Access to this field is **RO**.

Bits [15:12]

Reserved, RES0.

MTEFAR, bits [11:8]

Indicates whether FAR_ELx[63:60] are UNKNOWN on a synchronous exception due to a Tag Check Fault.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTEFAR	Meaning
0b0000	On a synchronous exception due to a Tag Check Fault, FAR_ELx[63:60] are UNKNOWN.
0b0001	On a synchronous exception due to a Tag Check Fault, FAR_ELx[63:60] are not UNKNOWN.

All other values are reserved.

FEAT_MTE_TAGGED_FAR implements the functionality identified by the value 0b0001.

If FEAT_MTE2 is not implemented, the value 0b0001 is not permitted.

From Armv8.9, if FEAT_MTE2 is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

MTESTOREONLY, bits [7:4]

Support for Store-only Tag checking.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTESTOREONLY	Meaning
0b0000	Store-only Tag checking is not supported.
0b0001	Store-only Tag checking is supported.

All other values are reserved.

FEAT_MTE_STORE_ONLY implements the functionality identified by the value 0b0001.

If FEAT_MTE2 is not implemented, the value 0b0001 is not permitted.

From Armv8.9, if FEAT_MTE2 is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

MTEPERM, bits [3:0]

Support for Allocation tag access permissions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTEPERM	Meaning
0b0000	Allocation tag access permissions are not supported.
0b0001	Allocation tag access permissions are supported. **Note** NoTagAccess is supported at stage 2 of translation only.

All other values are reserved.

FEAT_MTE_PERM implements the functionality identified by the value 0b0001

If FEAT_MTE2 is not implemented, the value 0b0001 is not permitted.

From Armv8.9, if FEAT_MTE2 is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

Accessing ID_AA64PFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64PFR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64PFR2_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_AA64PFR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64PFR2_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64SMFR0_EL1, SME Feature ID Register 0

The ID_AA64SMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented features of the AArch64 Scalable Matrix Extension.

The fields in this register do not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64SMFR0_EL1 and ID_AA64FPFR0_EL1.

Configuration

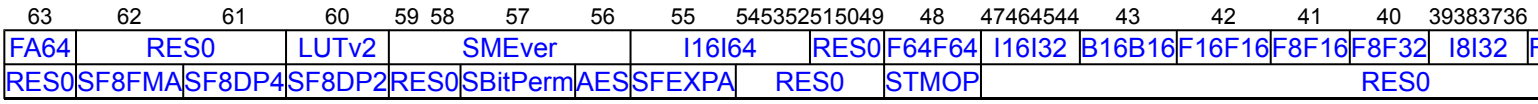
Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64SMFR0_EL1 is a 64-bit register.

Field descriptions



FA64, bit [63]

Indicates support at each Exception Level for execution of the full AArch64 Advanced SIMD and SVE instruction sets when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FA64	Meaning
0b0	Only those AArch64 instructions defined as being legal can be executed in streaming SVE mode.
0b1	All implemented AArch64 instructions are legal for execution in Streaming SVE mode, when enabled by SMCR_EL1.FA64, SMCR_EL2.FA64, and SMCR_EL3.FA64.

FEAT_SME_FA64 implements the functionality identified by the value 0b1.

Access to this field is **RO**.

Bits [62:61]

Reserved, RES0.

LUTv2, bit [60]

Indicates support for the following additional variants of SME2 lookup table LUTI4 and MOVT instructions:

- A LUTI4 instruction with 8-bit result elements, two consecutively numbered source vectors, and four consecutively numbered destination vectors.
- If FEAT_SME2p1 is implemented, a LUTI4 instruction with 8-bit result elements, two consecutively numbered source vectors, and four destination vectors with strided register numbering.
- A MOVT instruction that copies a single Z source vector to ZT0.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LUTv2	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

All other values are reserved.

FEAT_SME_LUTv2 implements the functionality identified by the value 0b1.

Access to this field is **RO**.

SMEver, bits [59:56]

Indicates support for SME instructions when FEAT_SME is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SMEver	Meaning
0b0000	The mandatory SME instructions are implemented.
0b0001	As 0b0000, and adds the mandatory SME2 instructions.
0b0010	As 0b0001, and adds the mandatory SME2.1 instructions.
0b0011	As 0b0010, and adds the mandatory SME2.2 instructions.

All other values are reserved.

FEAT_SME implements the functionality identified by the value 0b0000.

FEAT_SME2 implements the functionality identified by the value 0b0001.

FEAT_SME2p1 implements the functionality identified by the value 0b0010.

From Armv9.4, the value 0b0001 is not permitted.

FEAT_SME2p2 implements the functionality identified by 0b0011.

From Armv9.6, the values 0b0000 and 0b0010 are not permitted.

Access to this field is **RO**.

I16I64, bits [55:52]

Indicates support for the following SME instructions that accumulate into 64-bit integer elements in the ZA array:

- The variants of the ADDHA, ADDVA, SMOPA, SMOPS, SUMOPA, SUMOPS, UMOPA, UMOPS, USMOPA, and USMOPS instructions that accumulate into 64-bit integer tiles.
- When FEAT_SME2 is implemented, the variants of the ADD, ADDA, SDOT, SMLALL, SMLSLL, SUB, SUBA, SVDOT, UDOT, UMLALL, UMLSLL, and UVDOT instructions that accumulate into 64-bit integer elements in ZA array vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I16I64	Meaning
0b0000	The specified instructions are not implemented by this control.
0b1111	The specified instructions are implemented.

All other values are reserved.

FEAT_SME_I16I64 implements the functionality identified by the value 0b1111.

Access to this field is **RO**.

Bits [51:49]

Reserved, RES0.

F64F64, bit [48]

Indicates support for the following SME instructions that accumulate into double-precision floating-point elements in the ZA array:

- The variants of the FMOPA and FMOPS instructions that accumulate into double-precision tiles.
- When FEAT_SME2 is implemented, the variants of the FADD, FMLA, FMLS, and FSUB instructions that accumulate into double-precision elements in ZA array vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F64F64	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented .

FEAT_SME_F64F64 implements the functionality identified by the value 0b1.

Access to this field is **RO**.

I16I32, bits [47:44]

When FEAT_SME2 is implemented:

Indicates support for SME2 SMOPA (2-way), SMOPS (2-way), UMOPA (2-way), and UMOPS (2-way) instructions that accumulate 16-bit outer products into 32-bit integer tiles.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I16I32	Meaning
0b0000	The specified instructions are not implemented by this control.
0b0101	The specified instructions are implemented.

All other values are reserved.

If FEAT_SME2 is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

B16B16, bit [43]

Indicates support for the SME ZA-targeting non-widening BFloat16 BFADD, BFMLA, BFMLS, BFMOPA, BFMOPS, and BFSUB instructions with BFloat16 operands and results.

The value of this field is an IMPLEMENTATION DEFINED choice of:

B16B16	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

FEAT_SME_B16B16 implements the functionality identified by the value 0b1.

Access to this field is **RO**.

F16F16, bit [42]

Indicates support for the following SME2 half-precision floating-point instructions:

- FMOPA and FMOPS instructions that accumulate half-precision outer-products into half-precision tiles.
- Multi-vector FADD, FMLA, FMLS, and FSUB instructions with half-precision operands and results.
- Multi-vector FCVT and FCVTL instructions that convert half-precision inputs to single-precision results.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F16F16	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

FEAT_SME_F16F16 implements the functionality identified by the value 0b1.

Access to this field is **RO**.

F8F16, bit [41]

Indicates support for the following SME2 instructions:

- The ZA-targeting FP8 instructions FDOT (2-way), FMLAL, FMOPA (2-way), and FVDOT that accumulate into half-precision floating-point elements.
- ZA-targeting non-widening half-precision FADD and FSUB instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8F16	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

FEAT_SME_F8F16 implements the functionality identified by the value 0b1.

Access to this field is **RO**.

F8F32, bit [40]

Indicates support for the SME2 ZA-targeting FP8 FDOT (4-way), FMLALL, FMOPA (4-way), FVDOTB, and FVDOTT instructions that accumulate into single-precision floating-point elements.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8F32	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

FEAT_SME_F8F32 implements the functionality identified by the value 0b1.

Access to this field is **RO**.

I8I32, bits [39:36]

When FEAT_SME is implemented:

Indicates support for SME SMOPA, SMOPS, SUMOPA, SUMOPS, UMOPA, UMOPS, USMOPA, and USMOPS instructions that accumulate 8-bit outer products into 32-bit tiles

The value of this field is an IMPLEMENTATION DEFINED choice of:

I8I32	Meaning
0b0000	The specified instructions are not implemented by this control.
0b1111	The specified instructions are implemented.

All other values are reserved.

If FEAT_SME is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

F16F32, bit [35]
When FEAT_SME is implemented:

Indicates support for SME FMOPA and FMOPS instructions that accumulate half-precision floating-point outer products into single-precision floating-point tiles.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F16F32	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

If FEAT_SME is implemented, the value 0b0 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

B16F32, bit [34]
When FEAT_SME is implemented:

Indicates support for SME BFMOPA and BFMOPS instructions that accumulate BFloat16 outer products into single-precision floating-point tiles.

The value of this field is an IMPLEMENTATION DEFINED choice of:

B16F32	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

If FEAT_SME is implemented, the value 0b0 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

BI32I32, bit [33]
When FEAT_SME2 is implemented:

Indicates support for SME BMOPA and BMOPS instructions that accumulate thirty-two 1-bit binary outer products into 32-bit integer tiles.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BI32I32	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

If FEAT_SME2 is implemented, the value 0b0 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

F32F32, bit [32]

When FEAT_SME is implemented:

Indicates support for SME FMOPA and FMOPS instructions that accumulate single-precision floating-point outer products into single-precision floating-point tiles.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F32F32	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

If FEAT_SME is implemented, the value 0b0 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES0.

SF8FMA, bit [30]

Indicates support for the SVE2 FP8 to single-precision and half-precision multiply-accumulate FMLALB, FMLALT, FMLALLBB, FMLALLBT, FMLALLTB, and FMLALLTT instructions when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SF8FMA	Meaning
0b0	This control has no effect on Streaming SVE mode behavior.
0b1	The specified instructions are supported in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

If FEAT_SME2 and FEAT_FP8FMA are implemented, the value 0b0 is not permitted.

FEAT_SSVE_FP8FMA implements the functionality identified by the value 0b1.

Access to this field is **RO**.

SF8DP4, bit [29]

Indicates support for the SVE2 FP8 to single-precision 4-way dot product FDOT (4-way) instructions when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SF8DP4	Meaning
0b0	This control has no effect on Streaming SVE mode behavior.
0b1	The specified instructions are supported in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

If FEAT_SME2 and FEAT_FP8DOT4 are implemented, the value `0b0` is not permitted.

FEAT_SSVE_FP8DOT4 implements the functionality identified by the value `0b1`.

Access to this field is **RO**.

SF8DP2, bit [28]

Indicates support for the SVE2 FP8 to half-precision 2-way dot product `FDOT` (2-way) instructions when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SF8DP2	Meaning
`0b0`	This control has no effect on Streaming SVE mode behavior.
`0b1`	The specified instructions are supported in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

If FEAT_SME2 and FEAT_FP8DOT2 are implemented, the value `0b0` is not permitted.

FEAT_SSVE_FP8DOT2 implements the functionality identified by the value `0b1`.

Access to this field is **RO**.

Bits [27:26]

Reserved, RES0.

SBitPerm, bit [25]

Indicates support for the SVE bit permute instructions identified as implemented by ID_AA64ZFR0_EL1.BitPerm, when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SBitPerm	Meaning
`0b0`	This control has no effect on Streaming SVE mode behavior.
`0b1`	The specified instructions are implemented when the PE is in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

FEAT_SSVE_BitPerm implements the functionality identified by the value `0b1`.

Access to this field is **RO**.

AES, bit [24]

Indicates support for the SVE AES and 128-bit polynomial multiply long instructions identified as implemented by ID_AA64ZFR0_EL1.AES, when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AES	Meaning
`0b0`	This control has no effect on Streaming SVE mode behavior.
`0b1`	The specified instructions are supported when the PE is in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

FEAT_SSVE_AES implements the functionality identified by the value `0b1`.

Access to this field is **RO**.

SFEXPA, bit [23]

Indicates support for the SVE `FEXPA` instruction when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SFEXPA	Meaning
`0b0`	This control has no effect on Streaming SVE mode behavior.
`0b1`	The specified instruction is supported when the PE is in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

FEAT_SSVE_FEXPA implements the functionality identified by the value `0b1`.

Access to this field is **RO**.

Bits [22:17]

Reserved, RES0.

STMOP, bit [16]

Indicates support for the following SME Structured sparsity outer product instructions:

- `BFTMOPA` (non-widening), if FEAT_SME_B16B16 is implemented.
- `BFTMOPA` (widening, BF16 to FP32).
- `FTMOPA` (non-widening, FP16), if FEAT_SME_F16F16 is implemented.
- `FTMOPA` (non-widening, FP32).
- `FTMOPA` (widening, 2-way, FP16 to FP32).
- `FTMOPA` (widening, 2-way, FP8 to FP16), if FEAT_SME_F8F16 is implemented.
- `FTMOPA` (widening, 4-way, FP8 to FP32) if FEAT_SME_F8F32 is implemented.
- `STMOPA`, `SUTMOPA`, `USTMOPA`, `UTMOPA` (4-way, Int8 to Int32). `STMOPA`, `UTMOPA` (2-way, Int16 to Int32).

The value of this field is an IMPLEMENTATION DEFINED choice of:

STMOP	Meaning
`0b0`	The specified instructions are not implemented by this control.
`0b1`	The specified instructions are implemented.

FEAT_SME_TMOP implements the functionality identified by the value `0b1`.

Access to this field is **RO**.

Bits [15:1]

Reserved, RES0.

SMOP4, bit [0]

Indicates support for the following SME Quarter-tile outer product instructions:

- BFMOP4A, BFMOP4S (non-widening, BF16), if FEAT_SME_B16B16 is implemented.
- BFMOP4A, BFMOP4S (widening, 2-way, BF16 to FP32).
- FMOP4A, FMOP4S (non-widening, FP16), if FEAT_SME_F16F16 is implemented.
- FMOP4A, FMOP4S (non-widening, FP32).
- FMOP4A, FMOP4S (non-widening, FP64), if FEAT_SME_F64F64 is implemented.
- FMOP4A, FMOP4S (widening, 2-way, FP16 to FP32).
- FMOP4A(widening, 2-way, FP8 to FP16), if FEAT_SME_F8F16 is implemented.
- FMOP4A(widening, 4-way, FP8 to FP32) instruction, if FEAT_SME_F8F32 is implemented.
- SMOP4A, SMOP4S, SUMOP4A, SUMOP4S, UMOP4A, UMOP4S, USMOP4A, USMOP4S (4-way, Int8 to Int32).
- SMOP4A, SMOP4S, SUMOP4A, SUMOP4S, UMOP4A, UMOP4S, USMOP4A, USMOP4S (4-way, Int16 to Int64), if FEAT_SME_I16I64 is implemented.
- SMOP4A, SMOP4S, UMOP4A, UMOP4S (2-way, Int16 to Int32).

The value of this field is an IMPLEMENTATION DEFINED choice of:

SMOP4	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

FEAT_SME_MOP4 implements the functionality identified by the value 0b1.

Access to this field is **RO**.

Accessing ID_AA64SMFR0_EL1

This register is read-only and can be accessed from EL1 and higher.

This register is only accessible from the AArch64 state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64SMFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b101

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64SMFR0_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_AA64SMFR0_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64SMFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64SMFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64SMFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64ZFR0_EL1, SVE Feature ID Register 0

The ID_AA64ZFR0_EL1 characteristics are:

Purpose

Provides additional information about the implemented features of the AArch64 Scalable Vector Extension instruction set, when FEAT_SVE or FEAT_SME is implemented.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

If FEAT_SME is implemented and FEAT_SVE is not implemented, then SVE instructions can only be executed when the PE is in Streaming SVE mode and the instructions are legal for execution in Streaming SVE mode.

Attributes

ID_AA64ZFR0_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0				F64MM				F32MM				F16MM				I8MM				SM4				RES0				SHA3			
RES0				B16B16				BF16				BitPerm				EltPerm				RES0				AES				SVEver			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:60]

Reserved, RES0.

F64MM, bits [59:56]

Indicates support for the SVE double-precision floating-point matrix multiply-accumulate instruction `FMMLA`, the `LD1RO*` instructions, the 128-bit element variants of the SVE `TRN1`, `TRN2`, `UZP1`, `UZP2`, `ZIP1`, and `ZIP2` instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F64MM	Meaning
`0b0000`	This field does not indicate support for the specified instructions.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_F64MM implements the functionality identified by `0b0001`.

The instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is **RO**.

F32MM, bits [55:52]

Indicates support for the SVE single-precision floating-point matrix multiply-accumulate instruction FMMLA.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F32MM	Meaning
0b0000	This field does not indicate support for the specified instruction.
0b0001	The specified instruction is implemented.

All other values are reserved.

FEAT_F32MM implements the functionality identified by 0b0001.

The instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is **RO**.

F16MM, bits [51:48]

Indicates support for the SVE half-precision floating-point matrix multiply-accumulate to single-precision instruction FMMLA (widening, FP16 to FP32).

The value of this field is an IMPLEMENTATION DEFINED choice of:

F16MM	Meaning
0b0000	This field does not indicate support for the specified instruction.
0b0001	The specified instruction is implemented.

FEAT_SVE_F16F32MM implements the functionality identified by 0b0001.

The instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is **RO**.

I8MM, bits [47:44]

Indicates support for the following SVE 8-bit integer sum-of-products and accumulate to 32-bit integer instructions SMMLA, SUDOT, UMMLA, USMMLA, and USDOT.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I8MM	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_I8MM implements the functionality identified by 0b0001.

When Advanced SIMD and SVE are both implemented, this field must return the same value as ID_AA64ISAR1_EL1.I8MM.

From Armv8.6, if SVE is implemented, the value 0b0000 is not permitted.

The SVE SMMLA, UMMLA, and USMMLA instructions might not be legal when the PE is in Streaming SVE mode.

Access to this field is **RO**.

SM4, bits [43:40]

Indicates support for SVE SM4 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SM4	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	SVE SM4E and SM4EKEY instructions are implemented.

All other values are reserved.

FEAT_SVE_SM4 implements the functionality identified by 0b0001.

The instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is **RO**.

Bits [39:36]

Reserved, RES0.

SHA3, bits [35:32]

Indicates support for the SVE SHA3 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA3	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	SVE RAX1 instruction is implemented.

All other values are reserved.

FEAT_SVE_SHA3 implements the functionality identified by 0b0001.

If FEAT_SME2p1 is not implemented, then instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is **RO**.

Bits [31:28]

Reserved, RES0.

B16B16, bits [27:24]

Indicates support for SVE non-widening BFloat16 instructions and SME multi-vector Z-targeting non-widening BFloat16 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

B16B16	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	The following non-widening BFloat16 instructions are implemented: • SVE instructions: BFADD, BFCLAMP, BFMAX, BFMAXNM, BFMIN, BFMINNM, BFMLA, BFMLS, BFMUL, and BFSUB. • If FEAT_SME2 is implemented, SME multi-vector Z-targeting instructions: BFCLAMP, BFMAX, BFMAXNM, BFMIN, and BFMINNM.
0b0010	As 0b0001, and adds the following non-widening BFloat16 instructions: • SVE instruction BFSCALE. • If FEAT_SME2 is implemented, SME multi-vector Z-targeting instructions BFMUL and BFSCALE.

FEAT_SVE_B16B16 implements the functionality identified by 0b0001.

FEAT_SVE_BFSCALE implements the functionality identified by 0b0010.

Access to this field is **RO**.

BF16, bits [23:20]

Indicates support for SVE BFloat16 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BF16	Meaning
0b0000	SVE BFloat16 instructions are not implemented.
0b0001	SVE BFCVT, BFCVTNT, BFDOT, BFMLALB, BFMLALT, and BFMMLA instructions are implemented.
0b0010	As 0b0001, but the FPCR.EBF field is also supported.

All other values are reserved.

FEAT_BF16 implements the functionality identified by 0b0001.

FEAT_EBF16 implements the functionality identified by 0b0010.

This field must return the same value as ID_AA64ISAR1_EL1.BF16.

The SVE BFMMLA instructions might not be legal when the PE is in Streaming SVE mode.

From Armv8.6 and Armv9.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

BitPerm, bits [19:16]

Indicates support for the SVE bit permute instructions BDEP, BEXT, and BGRP.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BitPerm	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_SVE_BitPerm implements the functionality identified by 0b0001.

If FEAT_SSVE_BitPerm is not implemented, then instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is **RO**.

EltPerm, bits [15:12]

When FEAT_SVE2p2 is implemented or FEAT_SME2p2 is implemented:

If FEAT_SVE2p2 is implemented, the following SVE instructions are implemented when the PE is not in Streaming SVE mode:

- 8-bit and 16-bit element COMPACT.
- 8-bit, 16-bit, 32-bit, and 64-bit element EXPAND.

If FEAT_SME2p2 is implemented, the following SVE instructions are implemented when the PE is in Streaming SVE mode:

- 8-bit, 16-bit, 32-bit, and 64-bit element COMPACT and EXPAND.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EltPerm	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	The specified instructions are implemented.

If FEAT_SVE2p2 or FEAT_SME2p2 is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [11:8]

Reserved, RES0.

AES, bits [7:4]

Indicates support for SVE Advanced Encryption Standard instructions and 128-bit polynomial multiply long instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AES	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	The following instructions are implemented: • SVE single-vector AESD, AESE, AESIMC, and AESMC instructions. • The 128-bit destination element variant of the SVE single-vector PMULLB and PMULLT instructions.
0b0010	As 0b0001.
0b0011	As 0b0010, and adds the following SVE instructions: • Multi-vector AESD, AESDIMC, AESE and AESEMC instructions. • Multi-vector 128-bit destination element PMULL and PMLAL instructions.

All other values are reserved.

FEAT_SVE_AES implements the functionality identified by the value 0b0001.

FEAT_SVE_PMULL128 implements the functionality identified by the value 0b0010.

FEAT_SVE_AES2 implements the functionality identified by 0b0011.

If FEAT_SSVE_AES is not implemented, then instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is **RO**.

SVEver, bits [3:0]

Indicates support for SVE instructions when FEAT_SME or FEAT_SVE is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SVEver	Meaning
0b0000	The SVE instructions are implemented.
0b0001	As 0b0000, and adds the mandatory SVE2 instructions.
0b0010	As 0b0001, and adds the mandatory SVE2.1 instructions.
0b0011	As 0b0010, and adds the mandatory SVE2.2 instructions.

All other values are reserved.

FEAT_SVE2 implements the functionality identified by 0b0001 when the PE is not in Streaming SVE mode.

FEAT_SME implements the functionality identified by 0b0001 when the PE is in Streaming SVE mode.

FEAT_SME2p1 implements the functionality identified by 0b0010 when the PE is in Streaming SVE mode.

FEAT_SVE2p1 implements the functionality identified by 0b0010 when the PE is not in Streaming SVE mode.

FEAT_SVE2p2 implements the functionality identified by 0b0011 when the PE is not in Streaming SVE mode.

FEAT_SME2p2 implements the functionality identified by 0b0011 when the PE is in Streaming SVE mode.

From Armv9, if this register is present, the value `0b0000` is not permitted.

From Armv9.4, the value `0b0001` is not permitted.

From Armv9.6, the value `0b0010` is not permitted.

Access to this field is **RO**.

Accessing ID_AA64ZFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64ZFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b100

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64ZFR0_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_AA64ZFR0_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ZFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ZFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64ZFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0

The ID_AFR0_EL1 characteristics are:

Purpose

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch32 state.

Must be interpreted with the Main ID Register, MIDR_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_AFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_AFR0[31:0].

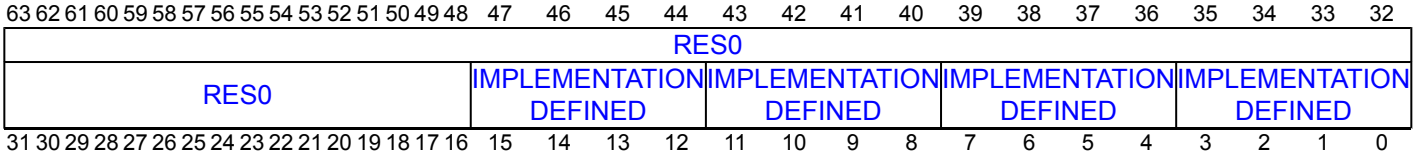
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AFR0_EL1 are UNDEFINED.

Attributes

ID_AFR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:16]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [15:12]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [11:8]

IMPLEMENTATION DEFINED.

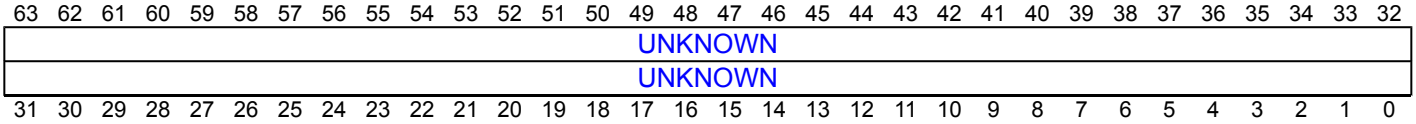
IMPLEMENTATION DEFINED, bits [7:4]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [3:0]

IMPLEMENTATION DEFINED.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_AFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b011

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AFR0_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_DFR0_EL1, AArch32 Debug Feature Register 0

The ID_DFR0_EL1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch32 state.

Must be interpreted with the Main ID Register, MIDR_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_DFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_DFR0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_DFR0_EL1 are UNDEFINED.

Attributes

ID_DFR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
TraceFilt				PerfMon				MProfDbg				MMapTrc				CopTrc				MMapDbg				CopSDbg				CopDbg			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:32]

Reserved, RES0.

TraceFilt, bits [31:28]

Armv8.4 Self-hosted Trace Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceFilt	Meaning
0b0000	Armv8.4 Self-hosted Trace Extension not implemented.
0b0001	Armv8.4 Self-hosted Trace Extension implemented.

All other values are reserved.

FEAT_TRF implements the functionality identified by the value 0b0001.

From Armv8.4, if FEAT_ETMv4 is implemented, the value 0b0000 is not permitted.

If FEAT_ETE is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

PerfMon, bits [27:24]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version'

The value of this field is an IMPLEMENTATION DEFINED choice of:

PerfMon	Meaning
0b0000	Performance Monitors Extension not implemented.
0b0001	Performance Monitors Extension, PMUv1 implemented.
0b0010	Performance Monitors Extension, PMUv2 implemented.
0b0011	Performance Monitors Extension, PMUv3 implemented.
0b0100	PMUv3 for Armv8.1. As 0b0011, and adds support for: • Extended 16-bit PMEVTYPER<n>.evtCount field. • If EL2 is implemented, the HDCR.HPMD control.
0b0101	PMUv3 for Armv8.4. As 0b0100, and adds support for the PMMIR register.
0b0110	PMUv3 for Armv8.5. As 0b0101, and adds support for: • 64-bit event counters. • If EL2 is implemented, the HDCR.HCCD control. • If EL3 is implemented, the MDCR_EL3.SCCD control.
0b0111	PMUv3 for Armv8.7. As 0b0110, and adds support for: • The PMCR.FZO and, if EL2 is implemented, HDCR.HPMFZO controls. • If EL3 is implemented, the MDCR_EL3.{MPMX,MCCD} controls.
0b1000	PMUv3 for Armv8.8. As 0b0111, and: • Extends the Common event number space to include 0x0040 to 0x00BF and 0x4040 to 0x40BF. • Removes the CONSTRAINED UNPREDICTABLE behaviors if a reserved or unimplemented PMU event number is selected.
0b1001	PMUv3 for Armv8.9. As 0b1000, and: • Updates the definitions of existing PMU events. • Adds support for the EDECR.PME control.
0b1111	IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value for new implementations.

All other values are reserved.

FEAT_PMUv3 implements the functionality identified by the value 0b0011.

FEAT_PMUv3p1 implements the functionality identified by the value 0b0100.

FEAT_PMUv3p4 implements the functionality identified by the value 0b0101.

FEAT_PMUv3p5 implements the functionality identified by the value 0b0110.

FEAT_PMUv3p7 implements the functionality identified by the value 0b0111.

FEAT_PMUv3p8 implements the functionality identified by the value 0b1000.

FEAT_PMUv3p9 implements the functionality identified by the value 0b1001.

In any Armv8 implementation, the values 0b0001 and 0b0010 are not permitted.

From Armv8.1, if FEAT_PMUv3 is implemented, the value 0b0011 is not permitted.

From Armv8.4, if FEAT_PMUv3 is implemented, the value 0b0100 is not permitted.

From Armv8.5, if FEAT_PMUv3 is implemented, the value 0b0101 is not permitted.

From Armv8.7, if FEAT_PMUv3 is implemented, the value 0b0110 is not permitted.

From Armv8.8, if FEAT_PMUv3 is implemented, the value 0b0111 is not permitted.

From Armv8.9, if FEAT_PMUv3 is implemented, the value 0b1000 is not permitted.

Access to this field is **RO**.

MProfDbg, bits [23:20]

M-profile Debug. Support for memory-mapped debug model for M-profile processors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MProfDbg	Meaning
0b0000	Not supported.
0b0001	Support for M-profile Debug architecture, with memory-mapped access.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

MMapTrc, bits [19:16]

Memory-mapped Trace. Support for memory-mapped trace model.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MMapTrc	Meaning
0b0000	Not supported.
0b0001	Support for Arm trace architecture, with memory-mapped access.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

For more information, see the Arm® Embedded Trace Macrocell Architecture Specification, ETMv4 (ARM IHI 0064).

Access to this field is **RO**.

CopTrc, bits [15:12]

Support for System registers-based trace model, using registers in the coproc == 0b1110 encoding space.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CopTrc	Meaning
0b0000	Not supported.
0b0001	Support for Arm trace architecture, with System registers access.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

For more information, see the Arm® Embedded Trace Macrocell Architecture Specification, ETMv4 (ARM IHI 0064).

Access to this field is **RO**.

MMapDbg, bits [11:8]

Memory-mapped Debug. Support for Armv7 memory-mapped debug model for A and R-profile processors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MMapDbg	Meaning
0b0000	Not supported.
0b0100	Support for Armv7, v7 Debug architecture, with memory-mapped access.
0b0101	Support for Armv7, v7.1 Debug architecture, with memory-mapped access.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

The optional memory map defined by Armv8 is not compatible with Armv7.

Access to this field is **RO**.

CopSDbg, bits [7:4]

Support for a System registers-based Secure debug model, using registers in the coproc = 0b1110 encoding space, for an A-profile processor that includes EL3.

If EL3 is not implemented and the implemented Security state is Non-secure state, this field is RES0. Otherwise, this field reads the same as bits [3:0].

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CopDbg, bits [3:0]

Debug architecture version. Indicates presence of Armv8 debug architecture.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CopDbg	Meaning
0b0000	Not supported.
0b0010	Armv6, v6 Debug architecture, with System registers access.
0b0011	Armv6, v6.1 Debug architecture, with System registers access.
0b0100	Armv7, v7 Debug architecture, with System registers access.
0b0101	Armv7, v7.1 Debug architecture, with System registers access.
0b0110	Armv8 debug architecture.
0b0111	Armv8.1 debug architecture, FEAT_Debugv8p1.
0b1000	Armv8.2 debug architecture, FEAT_Debugv8p2.
0b1001	Armv8.4 debug architecture, FEAT_Debugv8p4.
0b1010	Armv8.8 debug architecture, FEAT_Debugv8p8.
0b1011	Armv8.9 debug architecture, FEAT_Debugv8p9.

All other values are reserved.

The values 0b0000, 0b0010, 0b0011, 0b0100, and 0b0101 are not permitted in Armv8.

FEAT_Debugv8p1 implements the functionality identified by the value 0b0111.

FEAT_Debugv8p2 implements the functionality identified by the value 0b1000.

FEAT_Debugv8p4 implements the functionality identified by the value 0b1001.

FEAT_Debugv8p8 implements the functionality identified by the value 0b1010.

FEAT_Debugv8p9 implements the functionality identified by the value 0b1011.

From Armv8.1, when FEAT_Debugv8p1 is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

From Armv8.9, the value 0b1010 is not permitted.

Access to this field is **RO**.

Otherwise:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
UNKNOWN																															
UNKNOWN																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_DFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_DFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b010

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_DFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_DFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_DFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_DFR1_EL1, AArch32 Debug Feature Register 1

The ID_DFR1_EL1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch32.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_DFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_DFR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_DFR1_EL1 are UNDEFINED.

Note

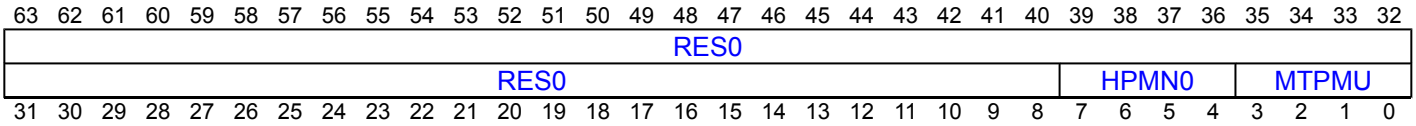
Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_DFR1_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:8]

Reserved, RES0.

HPMN0, bits [7:4]

Zero PMU event counters for a Guest operating system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HPMN0	Meaning
0b0000	Setting HDCR.HPMN to zero has CONSTRAINED UNPREDICTABLE behavior.
0b0001	Setting HDCR.HPMN to zero has defined behavior.

All other values are reserved.

If FEAT_PMUv3 is not implemented, FEAT_FGT is not implemented, or EL2 is not implemented, the only permitted value is 0b0000.

FEAT_HPMN0 implements the functionality identified by the value 0b0001.

From Armv8.8, in an implementation that includes FEAT_PMUv3, FEAT_FGT, and EL2, the value 0b0000 is not permitted.

Access to this field is **RO**.

MTPMU, bits [3:0]

Multi-threaded PMU extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTPMU	Meaning
0b0000	FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, it is IMPLEMENTATION DEFINED whether PMEVTYPER<n>.MT are read/write or RES0.
0b0001	FEAT_MTPMU and FEAT_PMUv3 implemented. PMEVTYPER<n>.MT are read/write. When FEAT_MTPMU is disabled, the Effective values of PMEVTYPER<n>.MT are 0.
0b1111	FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, PMEVTYPER<n>.MT are RES0.

All other values are reserved.

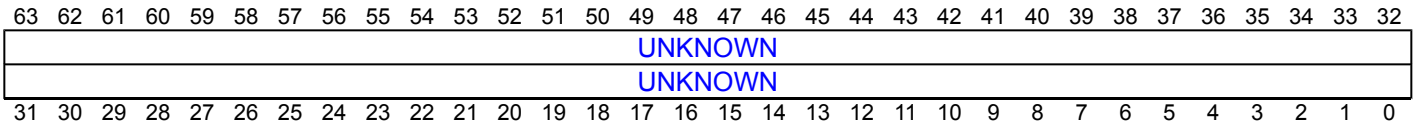
FEAT_MTPMU implements the functionality identified by the value 0b0001.

From Armv8.6, in an implementation that includes FEAT_PMUv3, the value 0b0000 is not permitted.

In an implementation that does not include FEAT_PMUv3, the value 0b0001 is not permitted.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_DFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_DFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b101

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_DFR1_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_DFR1_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_DFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_DFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_DFR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0

The ID_ISAR0_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR0[31:0].

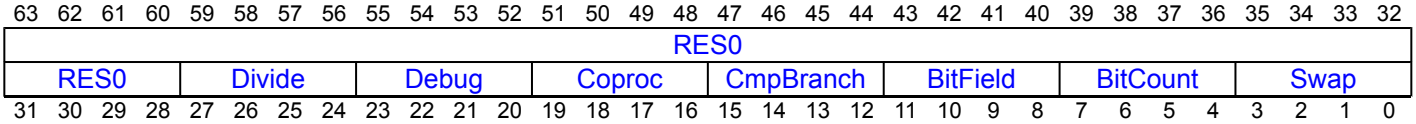
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR0_EL1 are UNDEFINED.

Attributes

ID_ISAR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:28]

Reserved, RES0.

Divide, bits [27:24]

Indicates the implemented Divide instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Divide	Meaning
0b0000	None implemented.
0b0001	Adds SDIV and UDIV in the T32 instruction set.
0b0010	As for 0b0001, and adds SDIV and UDIVin the A32 instruction set.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

Debug, bits [23:20]

Indicates the implemented Debug instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Debug	Meaning
0b0000	None implemented.
0b0001	Adds BKPT.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Coproc, bits [19:16]

Indicates the implemented System register access instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Coproc	Meaning
0b0000	None implemented, except for instructions separately attributed by the architecture to provide access to AArch32 System registers and System instructions.
0b0001	Adds generic CDP, LDC, MCR, MRC, and STC.
0b0010	As for 0b0001, and adds generic CDP2, LDC2, MCR2, MRC2, and STC2.
0b0011	As for 0b0010, and adds generic MCRR and MRRC.
0b0100	As for 0b0011, and adds generic MCRR2 and MRRC2.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

CmpBranch, bits [15:12]

Indicates the implemented combined Compare and Branch instructions in the T32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CmpBranch	Meaning
0b0000	None implemented.
0b0001	Adds CBNZ and CBZ.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

BitField, bits [11:8]

Indicates support for the following BitField instructions BFC, BFI, SBFX, and UBFX.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BitField	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

BitCount, bits [7:4]

Indicates the implemented Bit Counting instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BitCount	Meaning
0b0000	None implemented.
0b0001	Adds CLZ.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Swap, bits [3:0]

Indicates the implemented Swap instructions in the A32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

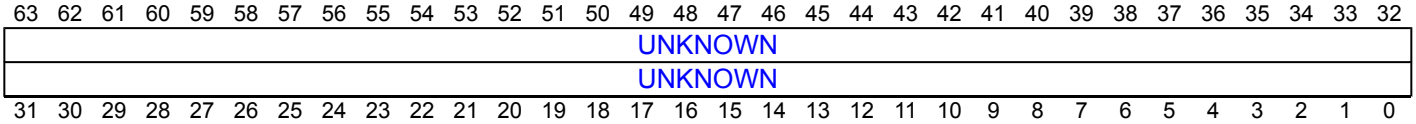
Swap	Meaning
0b0000	None implemented.
0b0001	Adds SWP and SWPB.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b000

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1

The ID_ISAR1_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR1[31:0].

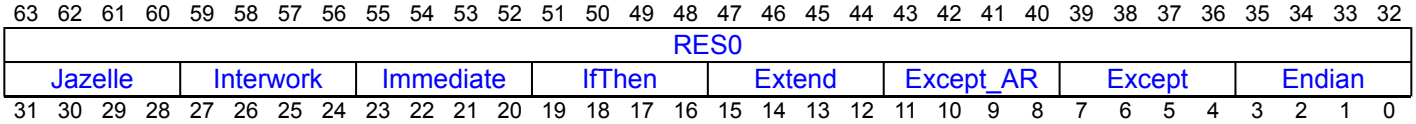
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR1_EL1 are UNDEFINED.

Attributes

ID_ISAR1_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

Jazelle, bits [31:28]

Indicates the implemented Jazelle extension instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Jazelle	Meaning
0b0000	No support for Jazelle.
0b0001	Adds the BXJ instruction and the J bit in the PSR. This setting might indicate a trivial implementation of the Jazelle extension.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Interwork, bits [27:24]

Indicates the implemented Interworking instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Interwork	Meaning
0b0000	None implemented.
0b0001	Adds the BX instruction, and the T bit in the PSR.
0b0010	As for 0b0001, and adds the BLX instruction. PC loads have BX-like behavior.
0b0011	As for 0b0010, and guarantees that data-processing instructions in the A32 instruction set with the PC as the destination and the S bit clear have BX-like behavior.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Access to this field is **RO**.

Immediate, bits [23:20]

Indicates support for the following data-processing instructions with long immediates:

- The MOVT instruction.
- The MOV instruction encodings with zero-extended 16-bit immediates.
- The T32 ADD and SUB instruction encodings with zero-extended 12-bit immediates, and the other ADD, ADR, and SUB encodings cross-referenced by the pseudocode for those encodings.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Immediate	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

IfThen, bits [19:16]

Indicates the implemented If-Then instructions in the T32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IfThen	Meaning
0b0000	None implemented.
0b0001	Adds the IT instructions, and the IT bits in the PSRs.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Extend, bits [15:12]

Indicates the implemented Extend instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Extend	Meaning
0b0000	No scalar sign-extend or zero-extend instructions are implemented, where scalar instructions means non-Advanced SIMD instructions.
0b0001	Adds the SXTB, SXTH, UXTB, and UXTH instructions.
0b0010	As for 0b0001, and adds the SXTB16, SXTAB, SXTAB16, SXTAH, UXTB16, UXTAB, UXTAB16, and UXTAH instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

Except_AR, bits [11:8]

Indicates the implemented A and R-profile exception-handling instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Except_AR	Meaning
0b0000	None implemented.
0b0001	Adds the SRS and RFE instructions, and the A and R-profile forms of the CPS instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Except, bits [7:4]

Indicates the implemented exception-handling instructions in the A32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Except	Meaning
0b0000	Not implemented. This indicates that the User bank and Exception return forms of the LDM and STM instructions are not implemented.
0b0001	Adds the LDM (exception return), LDM (user registers), and STM (user registers) instruction versions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Endian, bits [3:0]

Indicates the implemented Endian instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

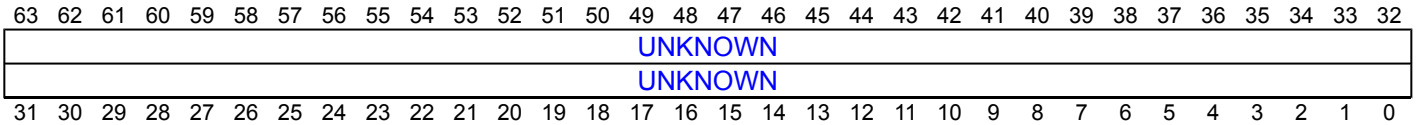
Endian	Meaning
0b0000	None implemented.
0b0001	Adds the SETEND instruction, and the E bit in the PSRs.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b001

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2

The ID_ISAR2_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR2[31:0].

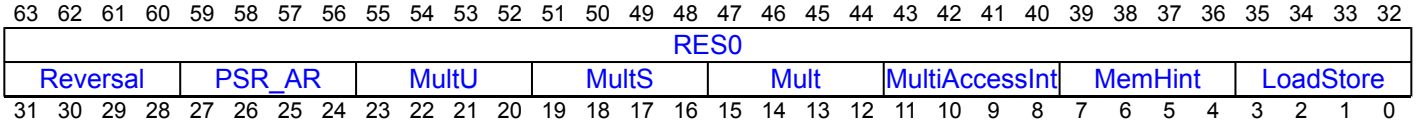
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR2_EL1 are UNDEFINED.

Attributes

ID_ISAR2_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

Reversal, bits [31:28]

Indicates the implemented Reversal instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Reversal	Meaning
0b0000	None implemented.
0b0001	Adds the REV, REV16, and REVSH instructions.
0b0010	As for 0b0001, and adds the RBIT instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

PSR_AR, bits [27:24]

Indicates the implemented A and R-profile instructions to manipulate the PSR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PSR_AR	Meaning
0b0000	None implemented.
0b0001	Adds the MRS and MSR instructions, and the exception return forms of data-processing instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

The exception return forms of the data-processing instructions are:

- In the A32 instruction set, data-processing instructions with the PC as the destination and the S bit set. These instructions might be affected by the WithShifts attribute.
- In the T32 instruction set, the SUBS PC,LR,#N instruction.

Access to this field is **RO**.

MultU, bits [23:20]

Indicates the implemented advanced unsigned Multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MultU	Meaning
0b0000	None implemented.
0b0001	Adds the UMULL and UMLAL instructions.
0b0010	As for 0b0001, and adds the UMAAL instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

MultS, bits [19:16]

Indicates the implemented advanced signed Multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MultS	Meaning
0b0000	None implemented.
0b0001	Adds the SMULL and SMLAL instructions.
0b0010	As for 0b0001, and adds the SMLABB, SMLABT, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, and SMULWT instructions. Also adds the Q bit in the PSRs.
0b0011	As for 0b0010, and adds the SMLAD, SMLADX, SMLALD, SMLALDX, SMLSD, SMLSDX, SMLSLD, SMLSLDX, SMMLA, SMMLAR, SMMLS, SMMLSR, SMMUL, SMMULR, SMUAD, SMUADX, SMUSD, and SMUSDX instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Access to this field is **RO**.

Mult, bits [15:12]

Indicates the implemented additional Multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Mult	Meaning
0b0000	No additional instructions implemented. This means only MUL is implemented.
0b0001	Adds the MLA instruction.
0b0010	As for 0b0001, and adds the MLS instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

MultiAccessInt, bits [11:8]

Indicates the support for interruptible multi-access instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MultiAccessInt	Meaning
0b0000	No support. This means the LDM and STM instructions are not interruptible.
0b0001	LDM and STM instructions are restartable.
0b0010	LDM and STM instructions are continuable.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

MemHint, bits [7:4]

Indicates the implemented Memory Hint instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MemHint	Meaning
0b0000	None implemented.
0b0001	Adds the PLD instruction.
0b0010	Adds the PLD instruction. (0b0001 and 0b0010 have identical effects.)
0b0011	As for 0b0001 (or 0b0010), and adds the PLI instruction.
0b0100	As for 0b0011, and adds the PLDW instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0100.

Access to this field is **RO**.

LoadStore, bits [3:0]

Indicates the implemented additional load/store instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

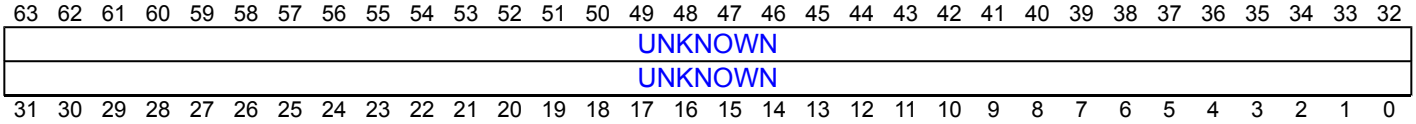
LoadStore	Meaning
0b0000	No additional load/store instructions implemented.
0b0001	Adds the LDRD and STRD instructions.
0b0010	As for 0b0001, and adds the Load Acquire (LDAB, LDAH, LDA, LDAEXB, LDAEXH, LDAEX, LDAEXD) and Store Release (STLB, STLH, STL, STLEXB, STLEXH, STLEX, STLEXD) instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b010

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR2_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3

The ID_ISAR3_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR3_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR3[31:0].

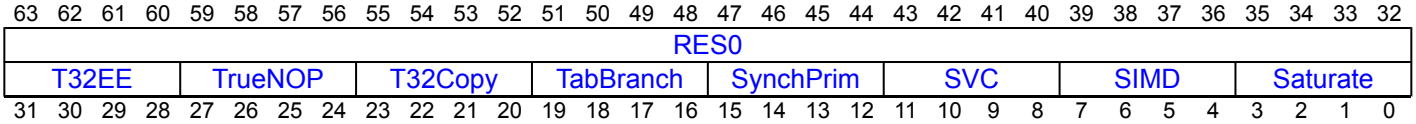
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR3_EL1 are UNDEFINED.

Attributes

ID_ISAR3_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

T32EE, bits [31:28]

Indicates the implemented T32EE instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

T32EE	Meaning
0b0000	None implemented.
0b0001	Adds the ENTERX and LEAVEX instructions, and modifies the load behavior to include null checking.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

TrueNOP, bits [27:24]

Indicates the implemented true NOP instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TrueNOP	Meaning
0b0000	None implemented. This means there are no NOP instructions that do not have any register dependencies.
0b0001	Adds true NOP instructions in both the T32 and A32 instruction sets. This also permits additional NOP-compatible hints.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

T32Copy, bits [23:20]

Indicates the support for T32 non flag-setting MOV instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

T32Copy	Meaning
0b0000	Not supported. This means that in the T32 instruction set, encoding T1 of the MOV (register) instruction does not support a copy from a low register to a low register.
0b0001	Adds support for T32 instruction set encoding T1 of the MOV (register) instruction, copying from a low register to a low register.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

TabBranch, bits [19:16]

Indicates the implemented Table Branch instructions in the T32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TabBranch	Meaning
0b0000	None implemented.
0b0001	Adds the TBB and TBH instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

SynchPrim, bits [15:12]

Used in conjunction with ID_ISAR4.SynchPrim_frac to indicate the implemented Synchronization Primitive instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SynchPrim	Meaning
0b0000	If SynchPrim_frac == 0b0000, no Synchronization Primitives implemented.
0b0001	If SynchPrim_frac == 0b0000, adds the LDREX and STREX instructions. If SynchPrim_frac == 0b0011, also adds the CLREX, LDREXB, STREXB, and STREXH instructions.
0b0010	If SynchPrim_frac == 0b0000, as for [0b0001, 0b0011] and also adds the LDREXD and STREXD instructions.

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

SVC, bits [11:8]

Indicates the implemented SVC instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SVC	Meaning
0b0000	Not implemented.
0b0001	Adds the SVC instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

SIMD, bits [7:4]

Indicates the implemented SIMD instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMD	Meaning
0b0000	None implemented.
0b0001	Adds the SSAT and USAT instructions, and the Q bit in the PSRs.
0b0011	As for 0b0001, and adds the PKHBT, PKHTB, QADD16, QADD8, QASX, QSUB16, QSUB8, QSAX, SADD16, SADD8, SASX, SEL, SHADD16, SHADD8, SHASX, SHSUB16, SHSUB8, SHSAX, SSAT16, SSUB16, SSUB8, SSAX, SXTAB16, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSUB16, UHSUB8, UHSAX, UQADD16, UQADD8, UQASX, UQSUB16, UQSUB8, UQSAX, USAD8, USADA8, USAT16, USUB16, USUB8, USAX, UXTAB16, and UXTB16 instructions. Also adds support for the GE[3:0] bits in the PSRs.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

The SIMD field relates only to implemented instructions that perform SIMD operations on the general-purpose registers. In an implementation that supports Advanced SIMD and floating-point instructions, MVFR0, MVFR1, and MVFR2 give information about the implemented Advanced SIMD instructions.

Access to this field is **RO**.

Saturate, bits [3:0]

Indicates the implemented Saturate instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

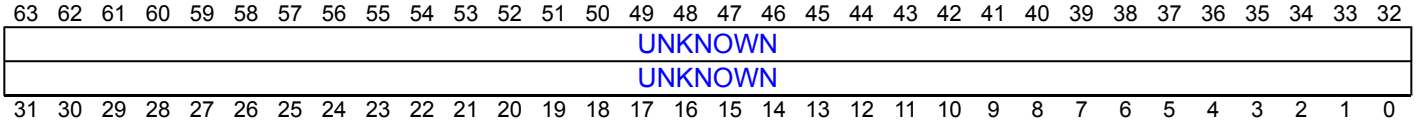
Saturate	Meaning
0b0000	None implemented. This means no non-Advanced SIMD saturate instructions are implemented.
0b0001	Adds the QADD, QDADD, QDSUB, and QSUB instructions, and the Q bit in the PSRs.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR3_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR3_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b011

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR3_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR3_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR3_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4

The ID_ISAR4_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR4_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR4[31:0].

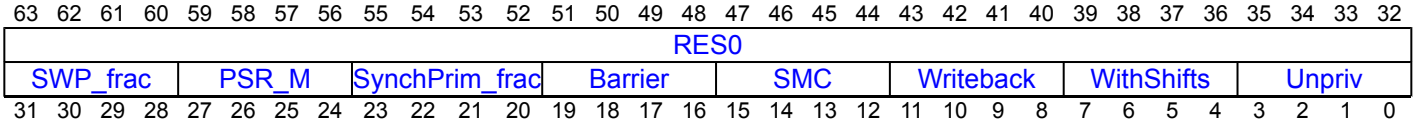
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR4_EL1 are UNDEFINED.

Attributes

ID_ISAR4_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

SWP_frac, bits [31:28]

Indicates support for the memory system locking the bus for SWP or SWPB instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SWP_frac	Meaning
0b0000	SWP or SWPB instructions not implemented.
0b0001	SWP or SWPB implemented but only in a uniprocessor context. SWP and SWPB do not guarantee whether memory accesses from other Requesters can come between the load memory access and the store memory access of the SWP or SWPB.

All other values are reserved. This field is valid only if ID_ISAR0.Swap is 0b0000.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

PSR_M, bits [27:24]

Indicates the implemented M-profile instructions to modify the PSRs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PSR_M	Meaning
0b0000	None implemented.
0b0001	Adds the M-profile forms of the CPS, MRS, and MSR instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

SynchPrim_frac, bits [23:20]

Used in conjunction with ID_ISAR3.SynchPrim to indicate the implemented Synchronization Primitive instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SynchPrim_frac	Meaning
0b0000	If SynchPrim == 0b0000, no Synchronization Primitives implemented. If SynchPrim == 0b0001, adds the LDREX and STREX instructions. If SynchPrim == 0b0010, also adds the CLREX, LDREXB, LDREXH, STREXB, STREXH, LDREXD, and STREXD instructions.
0b0011	If SynchPrim == 0b0001, adds the LDREX, STREX, CLREX, LDREXB, LDREXH, STREXB, and STREXH instructions.

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

Barrier, bits [19:16]

Indicates the implemented Barrier instructions in the T32 and A32 instruction sets.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Barrier	Meaning
0b0000	None implemented. Barrier operations are provided only as System instructions in the (coproc==0b1111) encoding space.
0b0001	Adds the DMB, DSB, and ISB barrier instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

SMC, bits [15:12]

Indicates the implemented SMC instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SMC	Meaning
0b0000	None implemented.
0b0001	Adds the SMC instruction.

All other values are reserved.

In Armv8-A, the permitted values are:

- If EL3 is implemented and EL1 can use AArch32, the only permitted value is 0b0001.
- If neither EL3 nor EL2 is implemented, the only permitted value is 0b0000.

If EL1 cannot use AArch32, this field has the value 0b0000.

Access to this field is **RO**.

Writeback, bits [11:8]

Indicates the support for Writeback addressing modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Writeback	Meaning
0b0000	Basic support. Only the LDM, STM, PUSH, POP, SRS, and RFE instructions support writeback addressing modes. These instructions support all of their writeback addressing modes.
0b0001	Adds support for all of the writeback addressing modes.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

WithShifts, bits [7:4]

Indicates the support for instructions with shifts.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WithShifts	Meaning
0b0000	Nonzero shifts supported only in MOV and shift instructions.
0b0001	Adds support for shifts of loads and stores over the range LSL 0-3.
0b0011	As for 0b0001, and adds support for other constant shift options, both on load/store and other instructions.
0b0100	As for 0b0011, and adds support for register-controlled shift options.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0100.

Access to this field is **RO**.

Unpriv, bits [3:0]

Indicates the implemented unprivileged instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

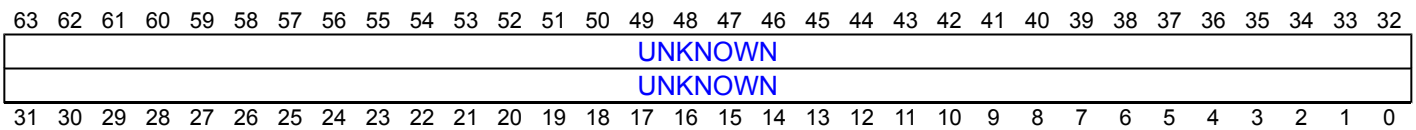
Unpriv	Meaning
0b0000	None implemented. No T variant instructions are implemented.
0b0001	Adds the LDRBT, LDRT, STRBT, and STRT instructions.
0b0010	As for 0b0001, and adds the LDRHT, LDRSBT, LDRSHT, and STRHT instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR4_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR4_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b100

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR4_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR4_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR4_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5

The ID_ISAR5_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, and ID_ISAR4_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR5_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR5[31:0].

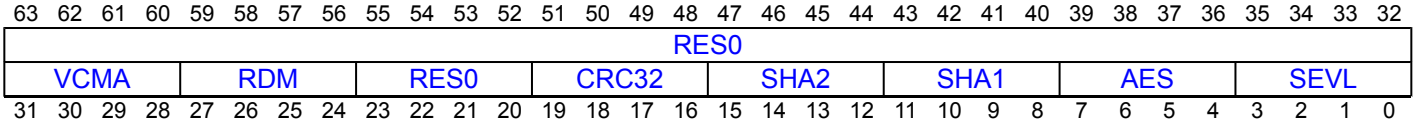
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR5_EL1 are UNDEFINED.

Attributes

ID_ISAR5_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

VCMA, bits [31:28]

Indicates AArch32 support for complex number addition and multiplication where numbers are stored in vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VCMA	Meaning
0b0000	The VCMLA and VCADD instructions are not implemented in AArch32.
0b0001	The VCMLA and VCADD instructions are implemented in AArch32.

All other values are reserved.

FEAT_FCMA implements the functionality identified by 0b0001.

From Armv8.3, the value 0b0000 is not permitted.

Access to this field is **RO**.

RDM, bits [27:24]

Indicates whether the VQRDMLAH and VQRDMLSH instructions are implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RDM	Meaning
0b0000	No VQRDMLAH and VQRDMLSH instructions implemented.
0b0001	VQRDMLAH and VQRDMLSH instructions implemented.

All other values are reserved.

FEAT_RDM implements the functionality identified by the value 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

CRC32, bits [19:16]

Indicates whether the CRC32 instructions CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, and CRC32CW are implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CRC32	Meaning
0b0000	CRC32 instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_CRC32 implements the functionality identified by the value 0b0001.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

SHA2, bits [15:12]

Indicates whether the SHA2 instructions SHA256H, SHA256H2, SHA256SU0, and SHA256SU1 are implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA2	Meaning
0b0000	No SHA2 instructions implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

SHA1, bits [11:8]

Indicates whether the SHA1 instructions SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 are implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA1	Meaning
0b0000	No SHA1 instructions implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

Access to this field is **RO**.

AES, bits [7:4]

Indicates whether the AES instructions are implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AES	Meaning
`0b0000`	No AES instructions implemented.
`0b0001`	`AESE`, `AESD`, `AESMC`, and `AESIMC` implemented.
`0b0010`	As for `0b0001`, plus `VMULL` (polynomial) instructions operating on 64-bit data quantities.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0010`.

Access to this field is **RO**.

SEVL, bits [3:0]

Indicates whether the `SEVL` instruction is implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

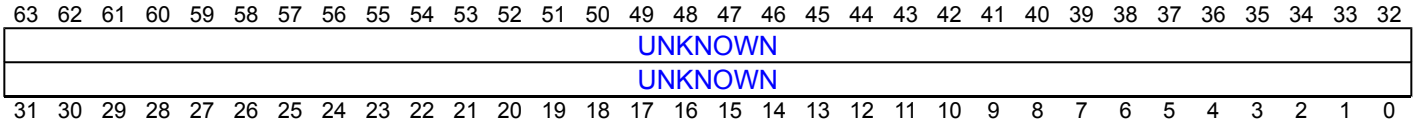
SEVL	Meaning
`0b0000`	`SEVL` is implemented as a NOP.
`0b0001`	`SEVL` is implemented as Send Event Local.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0001`.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR5_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR5_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b101

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR5_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR5_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR5_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6

The ID_ISAR6_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1 and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR6_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR6[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR6_EL1 are UNDEFINED.

Note

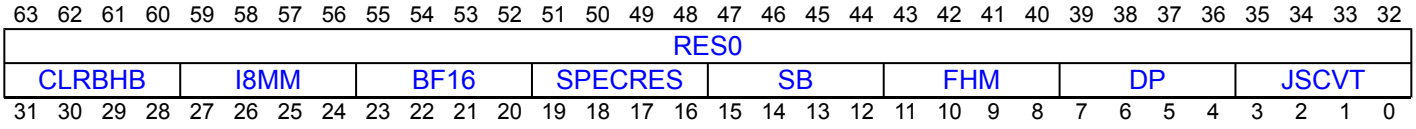
Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_ISAR6_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

CLRBHB, bits [31:28]

Indicates support for the CLRBHB instruction in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CLRBHB	Meaning
0b0000	CLRBHB instruction is not implemented.
0b0001	CLRBHB instruction is implemented.

All other values are reserved.

FEAT_CLRBHB implements the functionality identified by 0b0001.

From Armv8.9, the value 0b0000 is not permitted.

Access to this field is **RO**.

I8MM, bits [27:24]

Indicates support for the following Advanced SIMD Int8 matrix multiplication instructions VSMMLA, VSUDOT, VUMMLA, VUSMMLA, and VUSDOT in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I8MM	Meaning
0b0000	Int8 matrix multiplication instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_AA32I8MM implements the functionality identified by 0b0001.

Access to this field is **RO**.

BF16, bits [23:20]

Indicates support for the following Advanced SIMD and floating-point BFloat16 instructions VCVT, VCVTB, VCVTT, VDOT, VFMAB, VFMAT, and VMMLA instructions with BF16 operand or result types in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BF16	Meaning
0b0000	BFloat16 instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_AA32BF16 implements the functionality identified by 0b0001.

Access to this field is **RO**.

SPECRES, bits [19:16]

Indicates support for prediction invalidation instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPECRES	Meaning
0b0000	Prediction invalidation instructions are not implemented.
0b0001	CFPRCTX, DVPRCTX, and CPPRCTX instructions are implemented.
0b0010	As 0b0001, and COSPRCTX instruction is implemented.

All other values are reserved.

FEAT_SPECRES implements the functionality identified by 0b0001.

FEAT_SPECRES2 implements the functionality identified by 0b0010.

From Armv8.5, the value 0b0000 is not permitted.

From Armv8.9, the value 0b0001 is not permitted.

Access to this field is **RO**.

SB, bits [15:12]

Indicates support for the SB instruction in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SB	Meaning
0b0000	SB instruction is not implemented.
0b0001	SB instruction is implemented.

All other values are reserved.

FEAT_SB implements the functionality identified by 0b0001.

From Armv8.5, value 0b0000 is not permitted.

Access to this field is **RO**.

FHM, bits [11:8]

Indicates support for the following Advanced SIMD and floating-point VFMAL and VFMSL instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FHM	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_FHM implements the functionality identified by 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

DP, bits [7:4]

Indicates support for dot product instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DP	Meaning
0b0000	Dot product instructions are not implemented.
0b0001	VUDOT and VSDOT instructions are implemented.

All other values are reserved.

FEAT_DotProd implements the functionality identified by 0b0001.

In Armv8.2, the permitted values are 0b0000 and 0b0001.

From Armv8.4, the only permitted value is 0b0001.

Access to this field is **RO**.

JSCVT, bits [3:0]

Indicates support for the VJCVT instruction in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

JSCVT	Meaning
0b0000	The VJCVT instruction is not implemented.
0b0001	The VJCVT instruction is implemented.

All other values are reserved.

FEAT_JSCVT implements the functionality identified by 0b0001.

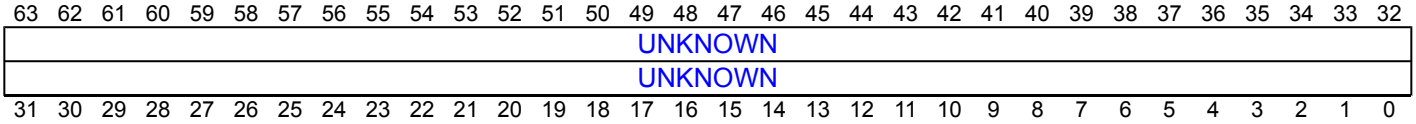
In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, if Advanced SIMD or Floating-point is implemented, the value 0b0000 is not implemented.

From Armv8.3, if Advanced SIMD or Floating-point is not implemented, the only permitted value is 0b0000.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR6_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR6_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b111

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_ISAR6_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_ISAR6_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR6_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR6_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR6_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0

The ID_MMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR0[31:0].

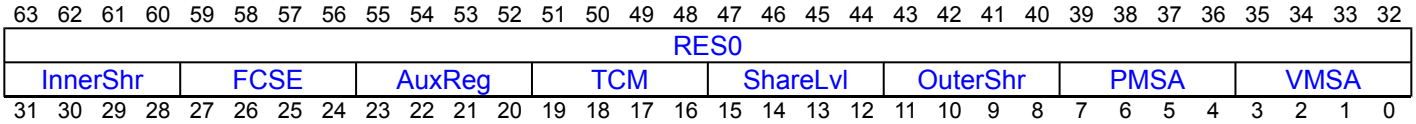
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR0_EL1 are UNDEFINED.

Attributes

ID_MMFR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

InnerShr, bits [31:28]

Innermost Shareability. Indicates the innermost shareability domain implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

InnerShr	Meaning
0b0000	Implemented as Non-cacheable.
0b0001	Implemented with hardware coherency support.
0b1111	Shareability ignored.

All other values are reserved.

From Armv8 the permitted values are 0b0000, 0b0001, and 0b1111.

This field is valid only if the implementation supports two levels of shareability, as indicated by ID_MMFR0_EL1.ShareLvl having the value 0b0001.

When ID_MMFR0_EL1.ShareLvl is zero, this field is UNKNOWN.

Access to this field is **RO**.

FCSE, bits [27:24]

Indicates whether the implementation includes the FCSE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FCSE	Meaning
0b0000	Not supported.
0b0001	Support for FCSE.

All other values are reserved.

From Armv8 the only permitted value is 0b0000.

Access to this field is **RO**.

AuxReg, bits [23:20]

Auxiliary Registers. Indicates support for Auxiliary registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AuxReg	Meaning
0b0000	None supported.
0b0001	Support for Auxiliary Control Register only.
0b0010	Support for Auxiliary Fault Status Registers (AIFSR and ADFSR) and Auxiliary Control Register.

All other values are reserved.

From Armv8 the only permitted value is 0b0010.

Note

Accesses to unimplemented Auxiliary registers are UNDEFINED.

Access to this field is **RO**.

TCM, bits [19:16]

Indicates support for TCMs and associated DMAs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TCM	Meaning
0b0000	Not supported.
0b0001	Support is IMPLEMENTATION DEFINED.
0b0010	Support for TCM only, Armv6 implementation.
0b0011	Support for TCM and DMA, Armv6 implementation.

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

Access to this field is **RO**.

ShareLvl, bits [15:12]

Shareability Levels. Indicates the number of shareability levels implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ShareLvl	Meaning
0b0000	One level of shareability implemented.
0b0001	Two levels of shareability implemented.

All other values are reserved.

From Armv8 the only permitted value is 0b0001.

Access to this field is **RO**.

OuterShr, bits [11:8]

Outermost Shareability. Indicates the outermost shareability domain implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OuterShr	Meaning
0b0000	Implemented as Non-cacheable.
0b0001	Implemented with hardware coherency support.
0b1111	Shareability ignored.

All other values are reserved.

From Armv8 the permitted values are 0b0000, 0b0001, and 0b1111.

Access to this field is **RO**.

PMSA, bits [7:4]

Indicates support for a PMSA.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMSA	Meaning
0b0000	Not supported.
0b0001	Support for IMPLEMENTATION DEFINED PMSA.
0b0010	Support for PMSAv6, with a Cache Type Register implemented.
0b0011	Support for PMSAv7, with support for memory subsections. Armv7-R profile.

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

Access to this field is **RO**.

VMSA, bits [3:0]

Indicates support for a VMSA.

The value of this field is an IMPLEMENTATION DEFINED choice of:

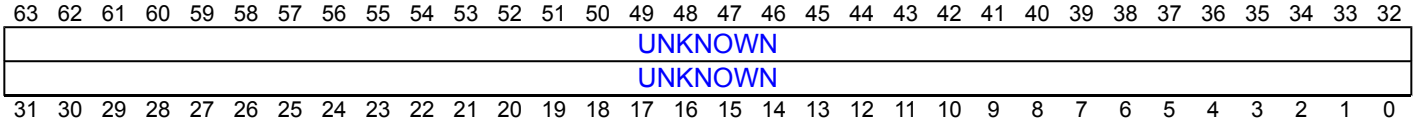
VMSA	Meaning
0b0000	Not supported.
0b0001	Support for IMPLEMENTATION DEFINED VMSA.
0b0010	Support for VMSAv6, with Cache and TLB Type Registers implemented.
0b0011	Support for VMSAv7, with support for remapping and the Access flag. Armv7-A profile.
0b0100	As for 0b0011, and adds support for the PXN bit in the Short-descriptor translation table format descriptors.
0b0101	As for 0b0100, and adds support for the Long-descriptor translation table format.

All other values are reserved.

In Armv8-A the only permitted value is 0b0101.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b100

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1

The ID_MMFR1_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR1[31:0].

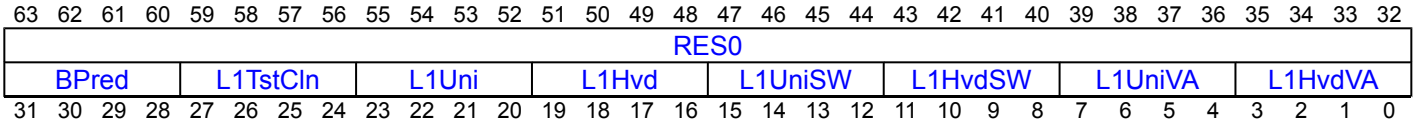
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR1_EL1 are UNDEFINED.

Attributes

ID_MMFR1_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

BPred, bits [31:28]

Branch Predictor. Indicates branch predictor management requirements.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BPred	Meaning
0b0000	No branch predictor, or no MMU present. Implies a fixed MPU configuration.
0b0001	Branch predictor requires flushing on: • Enabling or disabling a stage of address translation. • Writing new data to instruction locations. • Writing new mappings to the translation tables. • Changes to the TTBR0, TTBR1, or TTBCR registers. • Changes to the ContextID or ASID, or to the FCSE ProcessID if this is supported.
0b0010	Branch predictor requires flushing on: • Enabling or disabling a stage of address translation. • Writing new data to instruction locations. • Writing new mappings to the translation tables. • Any change to the TTBR0, TTBR1, or TTBCR registers without a change to the corresponding ContextID or ASID, or FCSE ProcessID if this is supported.
0b0011	Branch predictor requires flushing only on writing new data to instruction locations.
0b0100	For execution correctness, branch predictor requires no flushing at any time.

All other values are reserved.

In Armv8-A, the permitted values are 0b0010, 0b0011, and 0b0100. For values other than 0b0000 and 0b0100 the Arm Architecture Reference Manual, or the product documentation, might give more information about the required maintenance.

Access to this field is **RO**.

L1TstCln, bits [27:24]

Level 1 cache Test and Clean. Indicates the supported Level 1 data cache test and clean operations, for Harvard or unified cache implementations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1TstCln	Meaning
0b0000	None supported.
0b0001	Supported Level 1 data cache test and clean operations are: • Test and clean data cache.
0b0010	As for 0b0001, and adds: • Test, clean, and invalidate data cache.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1Uni, bits [23:20]

Level 1 Unified cache. Indicates the supported entire Level 1 cache maintenance operations for a unified cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1Uni	Meaning
0b0000	None supported.
0b0001	Supported entire Level 1 cache operations are: • Invalidate cache, including branch predictor if appropriate. • Invalidate branch predictor, if appropriate.
0b0010	As for 0b0001, and adds: • Clean cache, using a recursive model that uses the cache dirty status bit. • Clean and invalidate cache, using a recursive model that uses the cache dirty status bit.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1Hvd, bits [19:16]

Level 1 Harvard cache. Indicates the supported entire Level 1 cache maintenance operations for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1Hvd	Meaning
0b0000	None supported.
0b0001	Supported entire Level 1 cache operations are: • Invalidate instruction cache, including branch predictor if appropriate. • Invalidate branch predictor, if appropriate.
0b0010	As for 0b0001, and adds: • Invalidate data cache. • Invalidate data cache and instruction cache, including branch predictor if appropriate.
0b0011	As for 0b0010, and adds: • Clean data cache, using a recursive model that uses the cache dirty status bit. • Clean and invalidate data cache, using a recursive model that uses the cache dirty status bit.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1UniSW, bits [15:12]

Level 1 Unified cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a unified cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1UniSW	Meaning
0b0000	None supported.
0b0001	Supported Level 1 unified cache line maintenance operations by set/way are: • Clean cache line by set/way.
0b0010	As for 0b0001, and adds: • Clean and invalidate cache line by set/way.
0b0011	As for 0b0010, and adds: • Invalidate cache line by set/way.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1HvdSW, bits [11:8]

Level 1 Harvard cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdSW	Meaning
0b0000	None supported.
0b0001	Supported Level 1 Harvard cache line maintenance operations by set/way are: • Clean data cache line by set/way. • Clean and invalidate data cache line by set/way.
0b0010	As for 0b0001, and adds: • Invalidate data cache line by set/way.
0b0011	As for 0b0010, and adds: • Invalidate instruction cache line by set/way.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1UniVA, bits [7:4]

Level 1 Unified cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a unified cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1UniVA	Meaning
0b0000	None supported.
0b0001	Supported Level 1 unified cache line maintenance operations by VA are: • Clean cache line by VA. • Invalidate cache line by VA. • Clean and invalidate cache line by VA.
0b0010	As for 0b0001, and adds: • Invalidate branch predictor by VA, if branch predictor is implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1HvdVA, bits [3:0]

Level 1 Harvard cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdVA	Meaning
0b0000	None supported.
0b0001	Supported Level 1 Harvard cache line maintenance operations by VA are: • Clean data cache line by VA. • Invalidate data cache line by VA. • Clean and invalidate data cache line by VA. • Clean instruction cache line by VA.
0b0010	As for 0b0001, and adds: • Invalidate branch predictor by VA, if branch predictor is implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

Otherwise:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
UNKNOWN																															
UNKNOWN																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b101

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2

The ID_MMFR2_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR2[31:0].

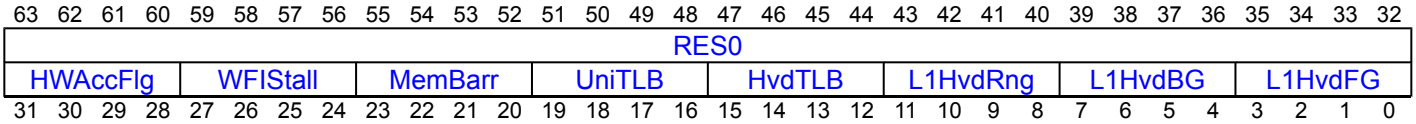
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR2_EL1 are UNDEFINED.

Attributes

ID_MMFR2_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

HWAccFlg, bits [31:28]

Hardware Access Flag. In earlier versions of the Arm Architecture, this field indicates support for a Hardware Access flag, as part of the VMSAv7 implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HWAccFlg	Meaning
0b0000	Not supported.
0b0001	Support for VMSAv7 Access flag, updated in hardware.

All other values are reserved.

From Armv8.0, 0b0001 is not permitted.

Access to this field is **RO**.

WFIStall, bits [27:24]

Wait For Interrupt Stall. Indicates the support for Wait For Interrupt (WFI) stalling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WFIStall	Meaning
0b0000	Not supported.
0b0001	Support for WFI stalling.

All other values are reserved.

Access to this field is **RO**.

MemBarr, bits [23:20]

Memory Barrier. Indicates the supported memory barrier System instructions in the (coproc==0b1111) encoding space:

The value of this field is an IMPLEMENTATION DEFINED choice of:

MemBarr	Meaning
0b0000	None supported.
0b0001	Supported memory barrier System instructions are: • Data Synchronization Barrier (DSB).
0b0010	As for 0b0001, and adds: • Instruction Synchronization Barrier (ISB). • Data Memory Barrier (DMB).

All other values are reserved.

From Armv8.0, the values 0b000 and 0b0001 are not permitted.

Arm deprecates the use of these operations. ID_ISAR4.Barrier_instrs indicates the level of support for the preferred barrier instructions.

Access to this field is **RO**.

UniTLB, bits [19:16]

Unified TLB. Indicates the supported TLB maintenance operations, for a unified TLB implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UniTLB	Meaning
0b0000	Not supported.
0b0001	Supported unified TLB maintenance operations are: • Invalidate all entries in the TLB. • Invalidate TLB entry by VA.
0b0010	As for 0b0001, and adds: • Invalidate TLB entries by ASID match.
0b0011	As for 0b0010, and adds: • Invalidate instruction TLB and data TLB entries by VA All ASID. This is a shared unified TLB operation.
0b0100	As for 0b0011, and adds: • Invalidate Hyp mode unified TLB entry by VA. • Invalidate entire Non-secure PL1&0 unified TLB. • Invalidate entire Hyp mode unified TLB.
0b0101	As for 0b0100, and adds the following operations: TLBIMVALIS, TLBIMVAALIS, TLBIMVALHIS, TLBIMVAL, TLBIMVAAL, TLBIMVALH.
0b0110	As for 0b0101, and adds the following operations: TLBIIPAS2IS, TLBIIPAS2LIS, TLBIIPAS2, TLBIIPAS2L.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0110.

Access to this field is **RO**.

HvdTLB, bits [15:12]

If the Unified TLB field (UniTLB, bits [19:16]) is not 0000, then the meaning of this field is IMPLEMENTATION DEFINED. Arm deprecates the use of this field by software.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

L1HvdRng, bits [11:8]

Level 1 Harvard cache Range. Indicates the supported Level 1 cache maintenance range operations, for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdRng	Meaning
0b0000	Not supported.
0b0001	Supported Level 1 Harvard cache maintenance range operations are: • Invalidate data cache range by VA. • Invalidate instruction cache range by VA. • Clean data cache range by VA. • Clean and invalidate data cache range by VA.

All other values are reserved.

From Armv8.0, the value 0b0001 is not permitted.

Access to this field is **RO**.

L1HvdBG, bits [7:4]

Level 1 Harvard cache Background fetch. Indicates the supported Level 1 cache background fetch operations, for a Harvard cache implementation. When supported, background fetch operations are non-blocking operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdBG	Meaning
0b0000	Not supported.
0b0001	Supported Level 1 Harvard cache background fetch operations are: • Fetch instruction cache range by VA. • Fetch data cache range by VA.

All other values are reserved.

From Armv8.0, the value 0b0001 is not permitted.

Access to this field is **RO**.

L1HvdFG, bits [3:0]

Level 1 Harvard cache Foreground fetch. Indicates the supported Level 1 cache foreground fetch operations, for a Harvard cache implementation. When supported, foreground fetch operations are blocking operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

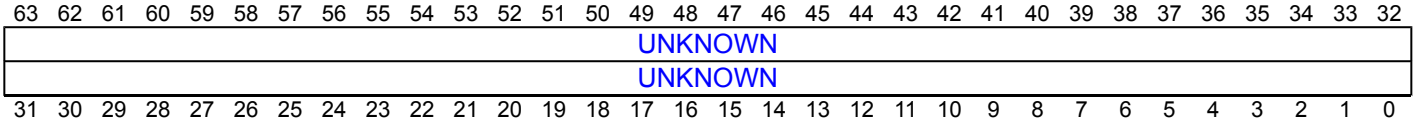
L1HvdFG	Meaning
0b0000	Not supported.
0b0001	Supported Level 1 Harvard cache foreground fetch operations are: • Fetch instruction cache range by VA. • Fetch data cache range by VA.

All other values are reserved.

From Armv8.0, the value 0b0001 is not permitted.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b110

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR2_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3

The ID_MMFR3_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR3_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR3[31:0].

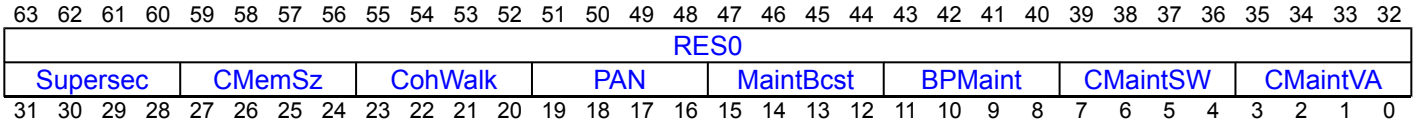
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR3_EL1 are UNDEFINED.

Attributes

ID_MMFR3_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

Supersec, bits [31:28]

Supersections. On a VMSA implementation, indicates whether Supersections are supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Supersec	Meaning
0b0000	Supersections supported.
0b1111	Supersections not supported.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b1111.

Access to this field is **RO**.

CMemSz, bits [27:24]

Cached Memory Size. Indicates the physical memory size supported by the caches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CMemSz	Meaning
0b0000	4GB, corresponding to a 32-bit physical address range.
0b0001	64GB, corresponding to a 36-bit physical address range.
0b0010	1TB or more, corresponding to a 40-bit or larger physical address range.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000, 0b0001, and 0b0010.

Access to this field is **RO**.

CohWalk, bits [23:20]

Coherent Walk. Indicates whether Translation table updates require a clean to the Point of Unification.:

The value of this field is an IMPLEMENTATION DEFINED choice of:

CohWalk	Meaning
0b0000	Updates to the translation tables require a clean to the Point of Unification to ensure visibility by subsequent translation table walks.
0b0001	Updates to the translation tables do not require a clean to the Point of Unification to ensure visibility by subsequent translation table walks.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

PAN, bits [19:16]

Privileged Access Never. Indicates support for the PAN bit in CPSR, SPSR, and DSPSR in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PAN	Meaning
0b0000	PAN not supported.
0b0001	PAN supported.
0b0010	PAN supported and ATS1CPRP and ATS1CPWP instructions supported.

All other values are reserved.

FEAT_PAN implements the functionality identified by the value 0b0001.

FEAT_PAN2 implements the functionality added by the value 0b0010.

From Armv8.1, the value 0b0000 is not permitted.

From Armv8.2, the value 0b0001 is not permitted.

Access to this field is **RO**.

MaintBcst, bits [15:12]

Maintenance Broadcast. Indicates whether Cache, TLB, and branch predictor operations are broadcast.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MaintBcst	Meaning
0b0000	Cache, TLB, and branch predictor operations only affect local structures.
0b0001	Cache and branch predictor operations affect structures according to shareability and defined behavior of instructions. TLB operations only affect local structures.
0b0010	Cache, TLB, and branch predictor operations affect structures according to shareability and defined behavior of instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

BPMaint, bits [11:8]

Branch Predictor Maintenance. Indicates the supported branch predictor maintenance operations in an implementation with hierarchical cache maintenance operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BPMaint	Meaning
0b0000	None supported.
0b0001	Supported branch predictor maintenance operations are: • Invalidate all branch predictors.
0b0010	As for 0b0001, and adds: • Invalidate branch predictors by VA.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

CMaintSW, bits [7:4]

Cache Maintenance by Set/Way. Indicates the supported cache maintenance operations by set/way, in an implementation with hierarchical caches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CMaintSW	Meaning
0b0000	None supported.
0b0001	Supported hierarchical cache maintenance instructions by set/way are: • Invalidate data cache by set/way. • Clean data cache by set/way. • Clean and invalidate data cache by set/way.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

In a unified cache implementation, the data cache maintenance operations apply to the unified caches.

Access to this field is **RO**.

CMaintVA, bits [3:0]

Cache Maintenance by Virtual Address. Indicates the supported cache maintenance operations by VA, in an implementation with hierarchical caches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CMaintVA	Meaning
0b0000	None supported.
0b0001	Supported hierarchical cache maintenance operations by VA are: • Invalidate data cache by VA. • Clean data cache by VA. • Clean and invalidate data cache by VA. • Invalidate instruction cache by VA. • Invalidate all instruction cache entries.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

In a unified cache implementation, data cache maintenance operations apply to the unified caches, and the instruction cache maintenance instructions are not implemented.

Access to this field is **RO**.

Otherwise:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
UNKNOWN																															
UNKNOWN																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR3_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR3_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b111

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR3_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR3_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR3_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4

The ID_MMFR4_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR4_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR4[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR4_EL1 are UNDEFINED.

Note

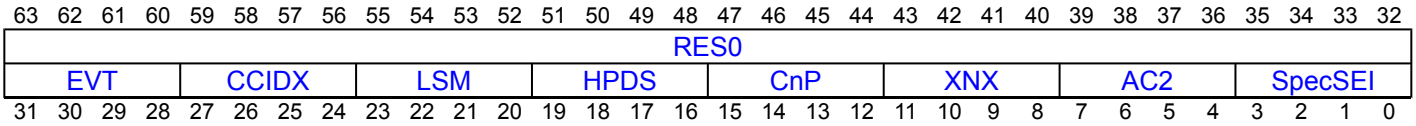
Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_MMFR4_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

EVT, bits [31:28]

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the HCR2.{TTLBIS, TOCU, TICAB, TID4} traps.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EVT	Meaning
0b0000	HCR2.{TTLBIS, TOCU, TICAB, TID4} traps are not supported.
0b0001	HCR2.{TOCU, TICAB, TID4} traps are supported. HCR2.TTLBIS trap is not supported.
0b0010	HCR2.{TTLBIS, TOCU, TICAB, TID4} traps are supported.

All other values are reserved.

FEAT_EVT implements the functionality identified by the values 0b0001 and 0b0010.

If EL2 is not implemented or does not support AArch32, the only permitted value is 0b0000.

From Armv8.5, if EL2 is supported and supports AArch32, the value 0b0001 is not permitted.

Access to this field is **RO**.

CCIDX, bits [27:24]

Support for use of the revised CCSIDR format and the presence of the CCSIDR2 is indicated.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCIDX	Meaning
0b0000	32-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is not implemented.
0b0001	64-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is implemented.

All other values are reserved.

FEAT_CCIDX implements the functionality identified by 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

LSM, bits [23:20]

Indicates support for LSMAOE and nTLSMD bits in HSCTLR and SCTLR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LSM	Meaning
0b0000	LSMAOE and nTLSMD bits not supported.
0b0001	LSMAOE and nTLSMD bits supported.

All other values are reserved.

FEAT_LSMAOC implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

HPDS, bits [19:16]

Hierarchical permission disables bits in translation tables.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HPDS	Meaning
0b0000	Disabling of hierarchical controls not supported.
0b0001	Supports disabling of hierarchical controls using the TTBCR2.HPD0, TTBCR2.HPD1, and HTCR.HPD bits.
0b0010	As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the Translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use.

All other values are reserved.

FEAT_AA32HPD implements the functionality identified by the value 0b0001.

FEAT_HPDS2 implements the functionality added by the value 0b0010.

Note

The value 0b0000 implies that the encoding for TTBCR2 is UNDEFINED.

Access to this field is **RO**.

CnP, bits [15:12]

Common not Private translations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CnP	Meaning
0b0000	Common not Private translations not supported.
0b0001	Common not Private translations supported.

All other values are reserved.

FEAT_TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2, the value 0b0000 is not permitted.

Access to this field is **RO**.

XNX, bits [11:8]

Support for execute-never control distinction by Exception level at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

XNX	Meaning
0b0000	Distinction between EL0 and EL1 execute-never control at stage 2 not supported.
0b0001	Distinction between EL0 and EL1 execute-never control at stage 2 supported.

All other values are reserved.

FEAT_XNX implements the functionality identified by the value 0b0001.

When FEAT_XNX is implemented:

- If all of the following conditions are true, it is IMPLEMENTATION DEFINED whether the value of ID_MMFR4_EL1.XNX is 0b0000 or 0b0001:
  - ID_AA64MMFR1_EL1.XNX ==1.
  - EL2 cannot use AArch32.
  - EL1 can use AArch32.
- If EL2 can use AArch32 then the value 0b0000 is not permitted.

Access to this field is **RO**.

AC2, bits [7:4]

Indicates the extension of the ACTLR and HACTLR registers using ACTLR2 and HACTLR2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AC2	Meaning
0b0000	ACTLR2 and HACTLR2 are not implemented.
0b0001	ACTLR2 and HACTLR2 are implemented.

All other values are reserved.

In Armv8.0 and Armv8.1 the permitted values are 0b0000 and 0b0001.

From Armv8.2, the only permitted value is 0b0001.

Access to this field is **RO**.

SpecSEI, bits [3:0]

When FEAT_RAS is implemented:

Describes whether the PE can generate SError exceptions from speculative reads of memory, including speculative instruction fetches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SpecSEI	Meaning
0b0000	The PE never generates an SError exception due to an External abort on a speculative read.
0b0001	The PE might generate an SError exception due to an External abort on a speculative read.

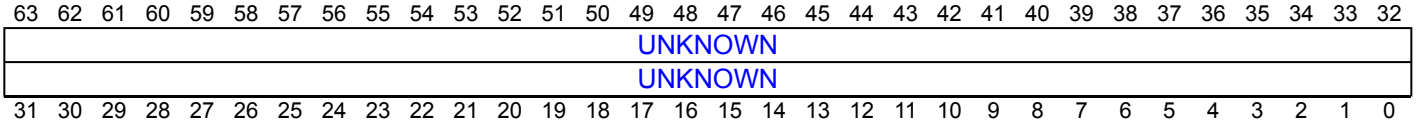
All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR4_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR4_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b110

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_MMFR4_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_MMFR4_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR4_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR4_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR4_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR5_EL1, AArch32 Memory Model Feature Register 5

The ID_MMFR5_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR5_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR5[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR5_EL1 are UNDEFINED.

Note

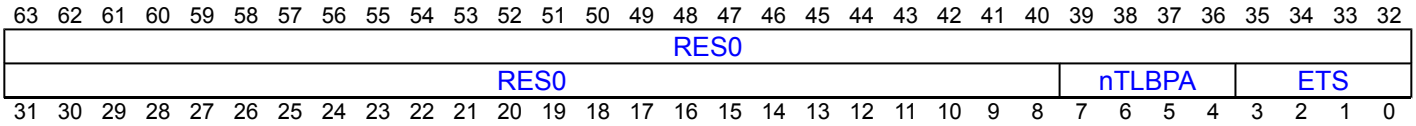
Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_MMFR5_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:8]

Reserved, RES0.

nTLBPA, bits [7:4]

Indicates support for intermediate caching of translation table walks.

The value of this field is an IMPLEMENTATION DEFINED choice of:

nTLBPA	Meaning
0b0000	The intermediate caching of translation table walks might include non-coherent physical translation caches.
0b0001	The intermediate caching of translation table walks does not include non-coherent physical translation caches.

Non-coherent physical translation caches are non-coherent caches of previous valid translation table entries since the last completed relevant TLBI applicable to the PE, where either:

- The caching is indexed by the physical address of the location holding the translation table entry.
- The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry.

All other values are reserved.

FEAT_nTLBPA implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

ETS, bits [3:0]

Indicates support for Enhanced Translation Synchronization.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ETS	Meaning
0b0000	Enhanced Translation Synchronization is not supported.
0b0001	Enhanced Translation Synchronization is not supported.
0b0010	FEAT_ETS2 is implemented.
0b0011	FEAT_ETS3 is implemented.

All other values are reserved.

FEAT_ETS2 implements the functionality identified by the value 0b0010.

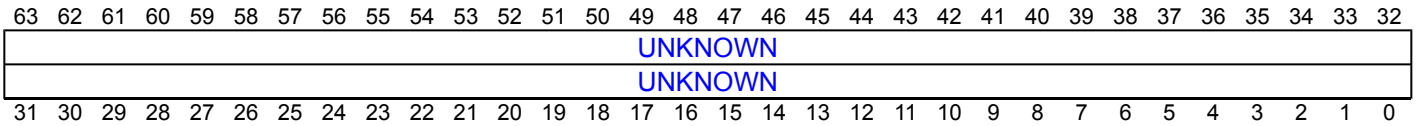
FEAT_ETS3 implements the functionality identified by the value 0b0011.

From Armv8.8, the values 0b0000 and 0b0001 are not permitted.

From Armv9.5, the value 0b0010 is not permitted.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR5_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR5_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b110

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_MMFR5_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_MMFR5_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR5_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR5_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR5_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_PFR0_EL1, AArch32 Processor Feature Register 0

The ID_PFR0_EL1 characteristics are:

Purpose

Gives top-level information about the instruction sets supported by the PE in AArch32 state.

Must be interpreted with ID_PFR1_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_PFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_PFR0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_PFR0_EL1 are UNDEFINED.

Attributes

ID_PFR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
RAS				DIT				AMU				CSV2				State3				State2				State1				State0			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:32]

Reserved, RES0.

RAS, bits [31:28]

RAS Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RAS	Meaning
0b0000	No RAS Extension.
0b0001	Support for the Reliability, Availability, and Serviceability Extension is implemented. The ESB instruction and the Error synchronization event are supported.
0b0010	As 0b0001, and adds support for additional ERXMISC<m> System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp Extension.
0b0011	As 0b0010, and requires that error records accessed through System registers conform to RAS System Architecture v2.

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0001.

FEAT_RASv1p1 implements the functionality identified by the value 0b0010.

FEAT_RASv2 implements the functionality identified by the value 0b0011.

In Armv8.0 and Armv8.1, the permitted values are 0b0000 and 0b0001.

From Armv8.2, the value 0b0000 is not permitted.

From Armv8.4, if FEAT_DoubleFault is implemented or ERRIDR_EL1.NUM is nonzero, the value 0b0001 is not permitted.

Note

When the value of this field is 0b0001, ID_PFR2_EL1.RAS_frac indicates whether FEAT_RASv1p1 is implemented.

From Armv8.9, if FEAT_DoubleFault is implemented or ERRIDR_EL1.NUM is nonzero, the value 0b0010 is not permitted.

Access to this field is **RO**.

DIT, bits [27:24]

Data Independent Timing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DIT	Meaning
0b0000	AArch32 does not guarantee constant execution time of any instructions.
0b0001	AArch32 provides the PSTATE.DIT mechanism to guarantee constant execution time of certain instructions.

All other values are reserved.

FEAT_DIT implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

Access to this field is **RO**.

AMU, bits [23:20]

Indicates support for Activity Monitors Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AMU	Meaning
0b0000	Activity Monitors Extension is not implemented.
0b0001	FEAT_AMUv1 is implemented.
0b0010	FEAT_AMUv1p1 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters.

All other values are reserved.

FEAT_AMUv1 implements the functionality identified by the value 0b0001.

FEAT_AMUv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0000.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.6, the permitted values are 0b0000, 0b0001, and 0b0010.

Access to this field is **RO**.

CSV2, bits [19:16]

Speculative use of out of context branch targets.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV2	Meaning
0b0000	The implementation does not disclose whether FEAT_CSV2 is implemented.
0b0001	FEAT_CSV2 is implemented, but FEAT_CSV2_1p1 is not implemented.
0b0010	FEAT_CSV2_1p1 is implemented.

All other values are reserved.

FEAT_CSV2 implements the functionality identified by the value 0b0001.

FEAT_CSV2_1p1 implements the functionality identified by the value 0b0010.

From Armv8.5, the permitted values are 0b0001 and 0b0010.

Access to this field is **RO**.

State3, bits [15:12]

T32EE instruction set support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State3	Meaning
0b0000	Not implemented.
0b0001	T32EE instruction set implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

State2, bits [11:8]

Jazelle extension support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State2	Meaning
0b0000	Not implemented.
0b0001	Jazelle extension implemented, without clearing of JOSCR.CV on exception entry.
0b0010	Jazelle extension implemented, with clearing of JOSCR.CV on exception entry.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

State1, bits [7:4]

T32 instruction set support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State1	Meaning
0b0000	T32 instruction set not implemented.
0b0001	T32 encodings before the introduction of Thumb-2 technology implemented: • All instructions are 16-bit. • A BL or BLX is a pair of 16-bit instructions. • 32-bit instructions other than BL and BLX cannot be encoded.
0b0011	T32 encodings after the introduction of Thumb-2 technology implemented, for all 16-bit and 32-bit T32 basic instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Access to this field is **RO**.

State0, bits [3:0]

A32 instruction set support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State0	Meaning
0b0000	A32 instruction set not implemented.
0b0001	A32 instruction set implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Otherwise:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
UNKNOWN																															
UNKNOWN																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_PFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_PFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b000

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_PFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_PFR1_EL1, AArch32 Processor Feature Register 1

The ID_PFR1_EL1 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with ID_PFR0_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_PFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_PFR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_PFR1_EL1 are UNDEFINED.

Attributes

ID_PFR1_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
GIC				Virt_frac				Sec_frac				GenTimer				Virtualization				MProgMod				Security				ProgMod			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:32]

Reserved, RES0.

GIC, bits [31:28]

System register GIC CPU interface.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GIC	Meaning
0b0000	GIC CPU interface system registers not implemented.
0b0001	System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.
0b0011	System register interface to version 4.1 of the GIC CPU interface is supported.

All other values are reserved.

Access to this field is **RO**.

Virt_frac, bits [27:24]

Virtualization fractional field. When the Virtualization field is 0b0000, determines the support for Virtualization Extensions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Virt_frac	Meaning
0b0000	No Virtualization Extensions are implemented.
0b0001	The following Virtualization Extensions are implemented: • The SCR.SIF bit, if EL3 is implemented. • The modifications to the SCR.AW and SCR.FW bits described in the Virtualization Extensions, if EL3 is implemented. • The MSR (banked register) and MRS (banked register) instructions. • The ERET instruction.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL2 is implemented.
- 0b0001 when EL2 is not implemented.

This field is valid only when the value of ID_PFR1_EL1.Virtualization is 0, otherwise it holds the value 0b0000.

Note

The ID_ISAR registers do not identify whether the instructions added by the Virtualization Extensions are implemented.

Access to this field is **RO**.

Sec_frac, bits [23:20]

Security fractional field. When the Security field is 0b0000, determines the support for Security Extensions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Sec_frac	Meaning
0b0000	No Security Extensions are implemented.
0b0001	The following Security Extensions are implemented: • The VBAR register. • The TTBCR.PD0 and TTBCR.PD1 bits.
0b0010	As for 0b0001, plus the ability to access Secure or Non-secure physical memory is supported.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL3 is implemented.
- 0b0001 or 0b0010 when EL3 is not implemented.

This field is valid only when the value of ID_PFR1_EL1.Security is 0, otherwise it holds the value 0b0000.

Access to this field is **RO**.

GenTimer, bits [19:16]

Generic Timer support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GenTimer	Meaning
0b0000	Generic Timer is not implemented.
0b0001	Generic Timer is implemented.
0b0010	Generic Timer is implemented, and also includes support for CNTHCTL.EVNTIS and CNTKCTL.EVNTIS fields, and CNTPCTSS and CNTVCTSS counter views.

All other values are reserved.

FEAT_ECV implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0001.

From Armv8.6, the only permitted value is 0b0010.

Access to this field is **RO**.

Virtualization, bits [15:12]

Virtualization support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Virtualization	Meaning
0b0000	EL2, Hyp mode, and the HVC instruction not implemented.
0b0001	EL2, Hyp mode, the HVC instruction, and all the features described by Virt_frac == 0b0001 implemented.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0001 when EL2 is implemented.

In an implementation that includes EL2, if EL2 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

If EL1 cannot use AArch32 then this field has the value 0b0000.

Note

The ID_ISARs do not identify whether the HVC instruction is implemented.

Access to this field is **RO**.

MProgMod, bits [11:8]

M-profile programmers' model support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MProgMod	Meaning
0b0000	Not supported.
0b0010	Support for two-stack programmers' model.

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

Access to this field is **RO**.

Security, bits [7:4]

Security support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Security	Meaning
0b0000	EL3, Monitor mode, and the SMC instruction not implemented.
0b0001	EL3, Monitor mode, the SMC instruction, and all the features described by Sec_frac == 0b0001 implemented.
0b0010	As for 0b0001, and adds the ability to set the NSACR.RFR bit. Not permitted in Armv8 as the NSACR.RFR bit is RES0.

All other values are reserved.

In Armv8-A, the permitted values are:

- `0b0000` when EL3 is not implemented.
- `0b0001` when EL3 is implemented.

In an implementation that includes EL3, if EL3 cannot use AArch32 but EL1 can use AArch32 then this field has the value `0b0001`.

If EL1 cannot use AArch32 then this field has the value `0b0000`.

Access to this field is **RO**.

ProgMod, bits [3:0]

Support for the standard programmers' model for Armv4 and later. Model must support User, FIQ, IRQ, Supervisor, Abort, Undefined, and System modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ProgMod	Meaning
`0b0000`	Not supported.
`0b0001`	Supported.

All other values are reserved.

In Armv8-A, the permitted values are `0b0001` and `0b0000`.

If EL1 cannot use AArch32 then this field has the value `0b0000`.

Access to this field is **RO**.

Otherwise:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
UNKNOWN																															
UNKNOWN																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_PFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_PFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b001

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_PFR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_PFR2_EL1, AArch32 Processor Feature Register 2

The ID_PFR2_EL1 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with ID_PFR0_EL1 and ID_PFR1_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_PFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_PFR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_PFR2_EL1 are UNDEFINED.

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_PFR2_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
RES0																				RAS_frac				SSBS				CSV3			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:12]

Reserved, RES0.

RAS_frac, bits [11:8]

RAS Extension fractional field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RAS_frac	Meaning
0b0000	If ID_PFR0_EL1.RAS == 0b0001, Support for the Reliability, Availability, and Serviceability Extension is implemented.
0b0001	If ID_PFR0_EL1.RAS == 0b0001, as 0b0000 and adds support for additional ERXMISC<m> System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp Extension.

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0000.

FEAT_RASv1p1 implements the functionality identified by the value 0b0001.

This field is valid only if ID_PFR0_EL1.RAS == 0b0001.

Access to this field is **RO**.

SSBS, bits [7:4]

Speculative Store Bypassing controls in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SSBS	Meaning
0b0000	AArch32 provides no mechanism to control the use of Speculative Store Bypassing.
0b0001	AArch32 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

All other values are reserved.

Access to this field is **RO**.

CSV3, bits [3:0]

Speculative use of faulting data.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV3	Meaning
0b0000	This PE does not disclose whether data loaded under speculation with a permission or domain fault can be used to form an address or generate condition codes or SVE predicate values to be used by other instructions in the speculative sequence.
0b0001	Data loaded under speculation with a permission or domain fault cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence. The execution timing of any other instructions in the speculative sequence is not a function of the data loaded under speculation.

All other values are reserved.

FEAT_CSV3 implements the functionality identified by the value 0b0001.

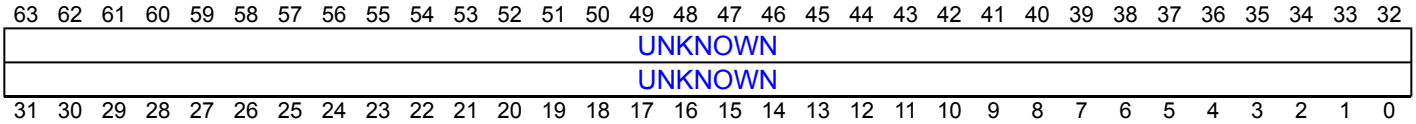
In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

If FEAT_E0PD is implemented, FEAT_CSV3 must be implemented.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_PFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_PFR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b100

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_PFR2_EL1) || boolean
IMPLEMENTATION_DEFINED "ID_PFR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_PFR2_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

IFSR32_EL2, Instruction Fault Status Register (EL2)

The IFSR32_EL2 characteristics are:

Purpose

Allows access to the AArch32 IFSR register from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register IFSR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register IFSR[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to IFSR32_EL2 are UNDEFINED.

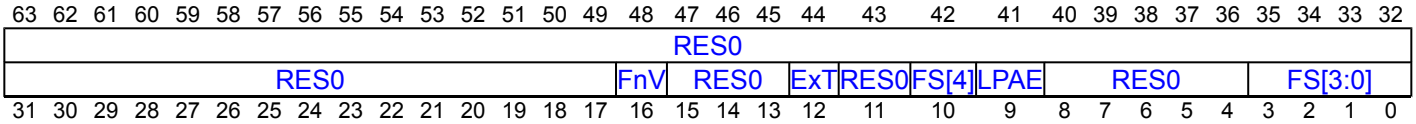
If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

Attributes

IFSR32_EL2 is a 64-bit register.

Field descriptions

When TTBCR.EAE == 0:



Bits [63:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	IFAR is valid.
0b1	IFAR is not valid, and holds an UNKNOWN value.

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault Status bits. Bits [10] and [3:0] are interpreted together.

FS	Meaning	Applies when
0b00001	PC alignment fault.	
0b00010	Debug exception.	
0b00011	Access flag fault, level 1.	
0b00101	Translation fault, level 1.	
0b00110	Access flag fault, level 2.	
0b00111	Translation fault, level 2.	
0b01000	Synchronous External abort, not on translation table walk.	
0b01001	Domain fault, level 1.	
0b01011	Domain fault, level 2.	
0b01100	Synchronous External abort, on translation table walk, level 1.	
0b01101	Permission fault, level 1.	
0b01110	Synchronous External abort, on translation table walk, level 2.	
0b01111	Permission fault, level 2.	
0b10000	TLB conflict abort.	
0b10100	IMPLEMENTATION DEFINED fault (Lockdown fault).	
0b11001	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b11100	Synchronous parity or ECC error on translation table walk, level 1.	When FEAT_RAS is not implemented
0b11110	Synchronous parity or ECC error on translation table walk, level 2.	When FEAT_RAS is not implemented

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Short-descriptor translation table lookup'.

The FS field is split as follows:

- FS[4] is IFSR32_EL2[10].
- FS[3:0] is IFSR32_EL2[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

LPAE	Meaning
0b0	Using the Short-descriptor translation table formats.
0b1	Using the Long-descriptor translation table formats.

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

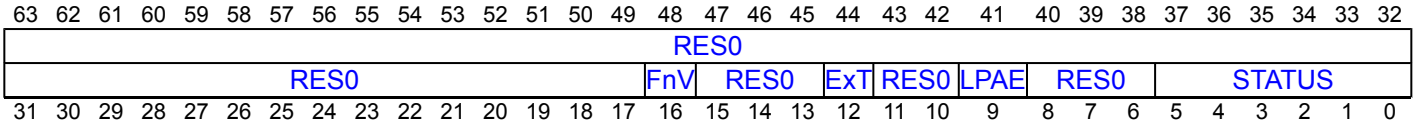
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When TTBCR.EAE == 1:



Bits [63:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	IFAR is valid.
0b1	IFAR is not valid, and holds an UNKNOWN value.

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

LPAE	Meaning
0b0	Using the Short-descriptor translation table formats.
0b1	Using the Long-descriptor translation table formats.

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status bits. Possible values of this field are:

STATUS	Meaning	Applies when
0b000000	Address size fault in translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk.	
0b010101	Synchronous External abort on translation table walk, level 1.	
0b010110	Synchronous External abort on translation table walk, level 2.	
0b010111	Synchronous External abort on translation table walk, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk, level 3.	When FEAT_RAS is not implemented
0b100001	PC alignment fault.	
0b100010	Debug exception.	
0b110000	TLB conflict abort.	

All other values are reserved.

When FEAT_RAS is implemented, 0b011000, 0b011101, 0b011110, and 0b011111 are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing IFSR32_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, IFSR32_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = IFSR32_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = IFSR32_EL2;
```

MSR IFSR32_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    IFSR32_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    IFSR32_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ISR_EL1, Interrupt Status Register

The ISR_EL1 characteristics are:

Purpose

Shows the pending status of IRQ and FIQ interrupts and SError exceptions.

When FEAT_NMI is implemented, also shows whether a pending IRQ or FIQ interrupt has Superpriority.

Configuration

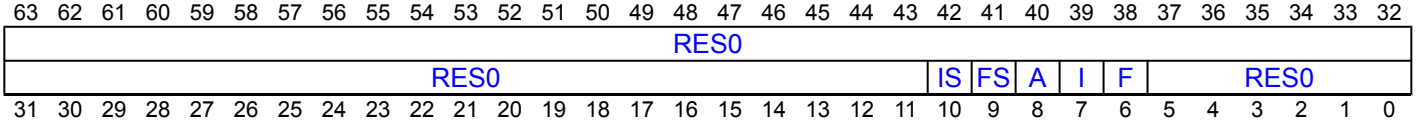
AArch64 System register ISR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ISR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ISR_EL1 are UNDEFINED.

Attributes

ISR_EL1 is a 64-bit register.

Field descriptions



Bits [63:11]

Reserved, RES0.

IS, bit [10]

When FEAT_NMI is implemented:

IRQ with Superpriority pending bit. Indicates whether an IRQ interrupt with Superpriority is pending.

IS	Meaning
0b0	No pending IRQ with Superpriority.
0b1	An IRQ interrupt with Superpriority is pending.

If all of the following apply then this field shows the pending status of virtual IRQ interrupts with Superpriority:

- EL2 is implemented and enabled in the current Security state.
- HCR_EL2.IMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical IRQ interrupts with Superpriority.

Otherwise:

Reserved, RES0.

FS, bit [9]

When FEAT_NMI is implemented:

FIQ with Superpriority pending bit. Indicates whether an FIQ interrupt with Superpriority is pending.

FS	Meaning
0b0	No pending FIQ with Superpriority.
0b1	An FIQ interrupt with Superpriority is pending.

If all of the following apply then this field shows the pending status of virtual FIQ interrupts with Superpriority:

- EL2 is implemented and enabled in the current Security state.
- HCR_EL2.FMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical FIQ interrupts with Superpriority.

Otherwise:

Reserved, RES0.

A, bit [8]

SError exception pending bit. Indicates whether an SError exception is pending.

A	Meaning
0b0	No pending SError.
0b1	An SError exception is pending.

If all of the following apply then this field shows the pending status of virtual SError exceptions:

- EL2 is implemented and enabled in the current Security state.
- Any of the following apply:
  - HCR_EL2.AMO is 1.
  - FEAT_DoubleFault2 is implemented and the Effective value of HCRX_EL2.TMEA is 1.
- The PE is executing at EL1.

Otherwise, if all of the following apply then this field shows the pending status of delegated SError exceptions:

- FEAT_E3DSE is implemented.
- SCR_EL3.EnDSE is 1.
- The PE is executing at EL2 or EL1.

Otherwise, this field shows the pending status of physical SError exceptions.

If the physical SError exception is edge-triggered, this field is cleared to zero when the physical SError exception is taken.

I, bit [7]

IRQ pending bit. Indicates whether an IRQ interrupt is pending.

I	Meaning
0b0	No pending IRQ.
0b1	An IRQ interrupt is pending.

If all of the following apply then this field shows the pending status of virtual IRQ interrupts:

- EL2 is implemented and enabled in the current Security state.
- HCR_EL2.IMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical IRQ interrupts.

Note

This bit indicates the presence of a pending IRQ interrupt regardless of whether the interrupt has Superpriority.

F, bit [6]

FIQ pending bit. Indicates whether an FIQ interrupt is pending.

F	Meaning
0b0	No pending FIQ.
0b1	An FIQ interrupt is pending.

If all of the following apply then this field shows the pending status of virtual FIQ interrupts:

- EL2 is implemented and enabled in the current Security state.
- HCR_EL2.FMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical FIQ interrupts.

Note

This bit indicates the presence of a pending FIQ interrupt regardless of whether the interrupt has Superpriority.

Bits [5:0]

Reserved, RES0.

Accessing ISR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ISR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.ISR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ISR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ISR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ISR_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

LORC_EL1, LORegion Control (EL1)

The LORC_EL1 characteristics are:

Purpose

Enables and disables LORegions, and selects the current LORegion descriptor.

Configuration

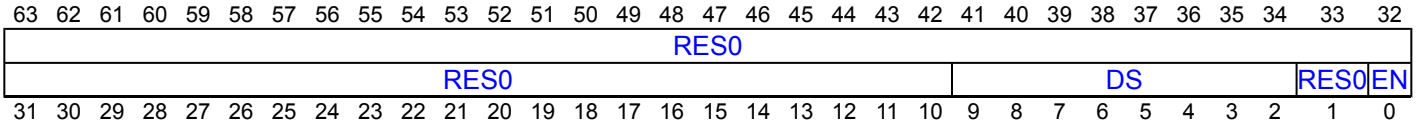
This register is present only when FEAT_LOR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to LORC_EL1 are UNDEFINED.

If no LORegion descriptors are supported by the PE, then this register is RES0.

Attributes

LORC_EL1 is a 64-bit register.

Field descriptions



Bits [63:10]

Reserved, RES0.

DS, bits [9:2]

Descriptor Select. Selects the current LORegion descriptor accessed by LORSA_EL1, LOREA_EL1, and LORN_EL1.

If this field points to an LORegion descriptor that is not supported by an implementation, then the registers LORN_EL1, LOREA_EL1, and LORSA_EL1 are RES0.

The number of LORegion descriptors in IMPLEMENTATION DEFINED. The maximum number of LORegion descriptors supported is 256. If the number is less than 256, then bits[63:M+2] are RES0, where M is Log_2(Number of LORegion descriptors supported by the implementation).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [1]

Reserved, RES0.

EN, bit [0]

Enable. Indicates whether LORegions are enabled.

EN	Meaning
0b0	Disabled. Memory accesses do not match any LORegions.
0b1	Enabled. Memory accesses may match a LORegion.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing LORC_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, LORC_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.LORC_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORC_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORC_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        X[t, 64] = LORC_EL1;
```

MSR LORC_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.LORC_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORC_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORC_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LORC_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

LOREA_EL1, LORegion End Address (EL1)

The LOREA_EL1 characteristics are:

Purpose

Holds the physical address of the end of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Configuration

This register is present only when FEAT_LOR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to LOREA_EL1 are UNDEFINED.

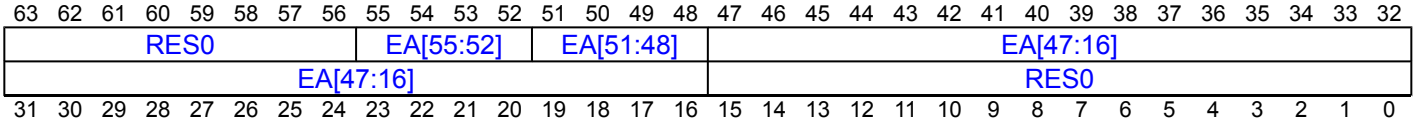
This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- LORC_EL1.DS points to a LORegion that is not supported by the PE.

Attributes

LOREA_EL1 is a 64-bit register.

Field descriptions



Any of the fields in this register are permitted to be cached in a TLB.

Bits [63:56]

Reserved, RES0.

EA[55:52], bits [55:52]

When FEAT_D128 is implemented:

Extension to EA[47:16]. For more information, see EA[47:16].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA[51:48], bits [51:48]

When FEAT_LPA is implemented:

Extension to EA[47:16]. For more information, see EA[47:16].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA[47:16], bits [47:16]

Bits [47:16] of the end physical address of an LORegion described in the current LORegion descriptor selected by LORC_EL1.DS. Bits[15:0] of this address are `0xFFFF`. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When FEAT_LPA is implemented and 52-bit addresses are in use, EA[51:48] form bits [51:48] of the end physical address of the LORegion. Otherwise, when 52-bit addresses are not in use, EA[51:48] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:0]

Reserved, RES0.

Accessing LOREA_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, LOREA_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.LOREA_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LOREA_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LOREA_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        X[t, 64] = LOREA_EL1;
```

```
MSR LOREA_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.LOREA_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LOREA_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LOREA_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LOREA_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

LORID_EL1, LORegionID (EL1)

The LORID_EL1 characteristics are:

Purpose

Indicates the number of LORegions and LORegion descriptors supported by the PE.

Configuration

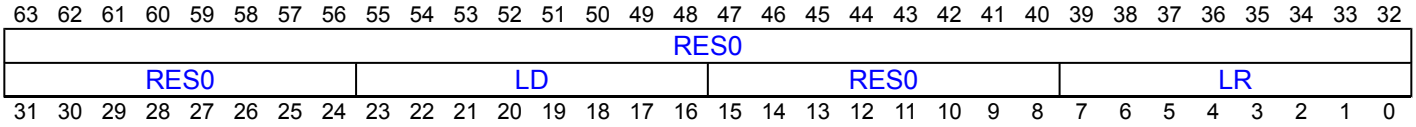
This register is present only when FEAT_LOR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to LORID_EL1 are UNDEFINED.

If no LORegion descriptors are implemented, then the registers LORC_EL1, LORN_EL1, LOREA_EL1, and LORSA_EL1 are RES0.

Attributes

LORID_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

LD, bits [23:16]

Number of LORegion descriptors supported by the PE. This is an 8-bit binary number.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LD	Meaning
0x00..0xFF	The number of LORegions descriptors supported.

Access to this field is **RO**.

Bits [15:8]

Reserved, RES0.

LR, bits [7:0]

Number of LORegions supported by the PE. This is an 8-bit binary number.

Note

If LORID_EL1 indicates that no LORegions are implemented, then LoadLOAcquire and StoreLORelease will behave as LoadAcquire and StoreRelease.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LR	Meaning
0x00..0xFF	The number of LORegions supported.

Access to this field is **RO**.

Accessing LORID_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, LORID_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.LORID_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORID_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORID_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = LORID_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

LORN_EL1, LORegion Number (EL1)

The LORN_EL1 characteristics are:

Purpose

Holds the number of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Configuration

This register is present only when FEAT_LOR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to LORN_EL1 are UNDEFINED.

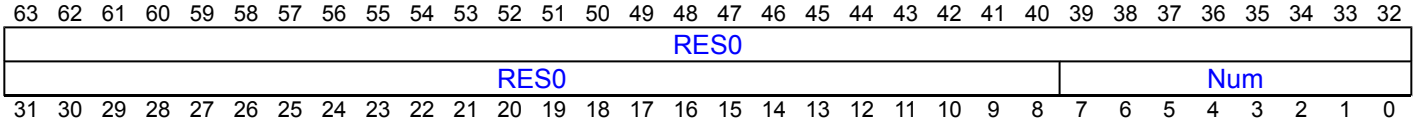
This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- LORC_EL1.DS points to a LORegion that is not supported by the PE.

Attributes

LORN_EL1 is a 64-bit register.

Field descriptions



Any of the fields in this register are permitted to be cached in a TLB.

Bits [63:8]

Reserved, RES0.

Num, bits [7:0]

Number of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

The maximum number of LORegions supported by the PE is 256. If the maximum number is less than 256, then bits[8:N] are RES0, where N is (Log_2(Number of LORegions supported by the PE)).

If this field points to a LORegion that is not supported by the PE, then the current LORegion descriptor does not match any LORegion.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing LORN_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, LORN_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.LORN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORN_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORN_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        X[t, 64] = LORN_EL1;
```

MSR LORN_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.LORN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORN_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORN_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LORN_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

LORSA_EL1, LORegion Start Address (EL1)

The LORSA_EL1 characteristics are:

Purpose

Indicates whether the current LORegion descriptor selected by LORC_EL1.DS is enabled, and holds the physical address of the start of the LORegion.

Configuration

This register is present only when FEAT_LOR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to LORSA_EL1 are UNDEFINED.

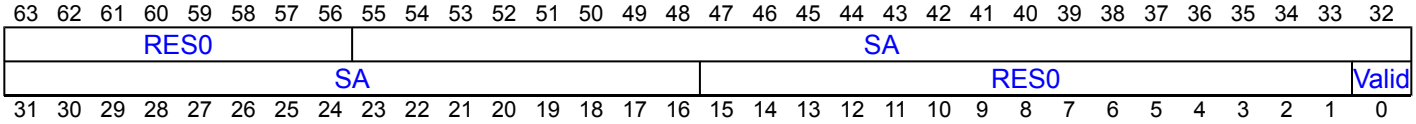
This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- LORC_EL1.DS points to a LORegion that is not supported by the PE.

Attributes

LORSA_EL1 is a 64-bit register.

Field descriptions



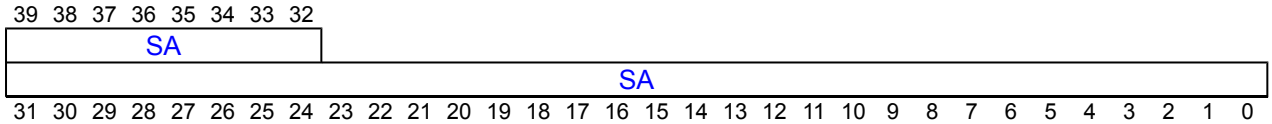
Any of the fields in this register are permitted to be cached in a TLB.

Bits [63:56]

Reserved, RES0.

SA, bits [55:16]

SA encoding when FEAT_D128 is implemented



SA, bits [39:0]

Bits [55:16] of the start physical address of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

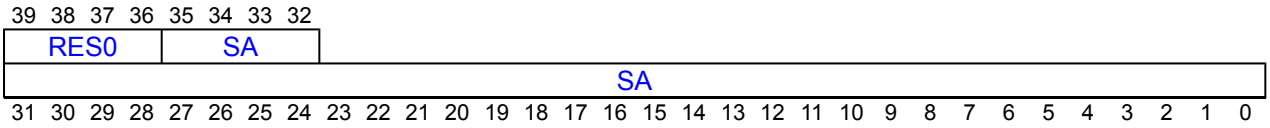
Bits[15:0] of this address are `0x0000`.

For implementations with fewer than 56 physical address bits, the corresponding upper bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SA encoding when FEAT_LPA is implemented and FEAT_D128 is not implemented



Bits [39:36]

Reserved, RES0.

SA, bits [35:0]

Bits [51:16] of the start physical address of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

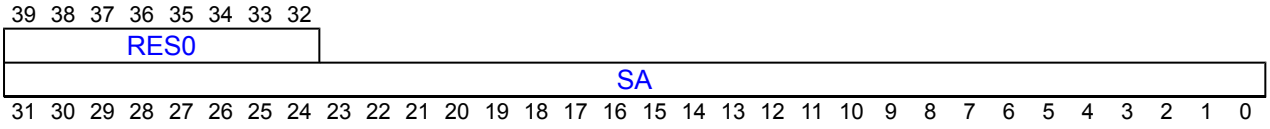
Bits[15:0] of this address are `0x0000`.

For implementations with fewer than 52 physical address bits, the corresponding upper bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SA encoding when FEAT_LPA is not implemented



Bits [39:32]

Reserved, RES0.

SA, bits [31:0]

Bits [47:16] of the start physical address of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Bits[15:0] of this address are `0x0000`.

For implementations with fewer than 48 physical address bits, the corresponding upper bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:1]

Reserved, RES0.

Valid, bit [0]

Indicates whether the current LORegion descriptor is enabled.

Valid	Meaning
`0b0`	LORegion descriptor is disabled.
`0b1`	LORegion descriptor is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing LORSA_EL1

Accesses to this register use the following encodings in the System register encoding space:

`MRS <Xt>, LORSA_EL1`

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.LORSA_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORSA_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORSA_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        X[t, 64] = LORSA_EL1;
```

`MSR LORSA_EL1, <Xt>`

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.LORSA_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORSA_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORSA_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LORSA_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR2_EL1, Extended Memory Attribute Indirection Register (EL1)

The MAIR2_EL1 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a VMSAv8-64 or VMSAv9-128 translation table entry for stage 1 translations at EL1.

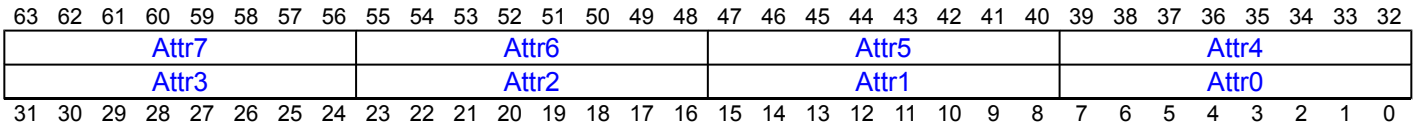
Configuration

This register is present only when FEAT_AIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR2_EL1 are UNDEFINED.

Attributes

MAIR2_EL1 is a 64-bit register.

Field descriptions



Attr<n>, bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 1, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 0, see MAIR_EL1.Attr

Attr is encoded as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of 'dd' for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise,UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise,UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR2_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name MAIR2_EL1 or MAIR2_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MAIR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x280];
    else
        X[t, 64] = MAIR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = MAIR2_EL2;
    else
        X[t, 64] = MAIR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR2_EL1;
```

```
MSR MAIR2_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x280] = X[t, 64];
    else
        MAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MAIR2_EL2 = X[t, 64];
    else
        MAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR2_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, MAIR2_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x280];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = MAIR2_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = MAIR2_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR MAIR2_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x280] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MAIR2_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        MAIR2_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR2_EL2, Extended Memory Attribute Indirection Register (EL2)

The MAIR2_EL2 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a VMSAv8-64 or VMSAv9-128 translation table entry for stage 1 translations at EL1.

Configuration

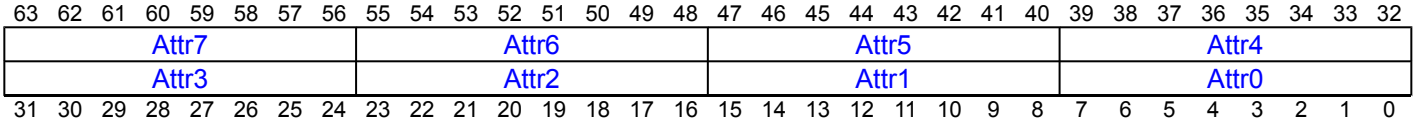
This register is present only when FEAT_AIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR2_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

MAIR2_EL2 is a 64-bit register.

Field descriptions



Attr<n>, bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 1, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 0, see MAIR_EL2.Attr

Attr is encoded as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of 'dd' for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise,UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise,UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name MAIR2_EL2 or MAIR2_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MAIR2_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MAIR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR2_EL2;
```

MSR MAIR2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MAIR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR2_EL2 = X[t, 64];
```

MRS <Xt>, MAIR2_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x280];
    else
        X[t, 64] = MAIR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = MAIR2_EL2;
    else
        X[t, 64] = MAIR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR2_EL1;
```

```
MSR MAIR2_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x280] = X[t, 64];
    else
        MAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MAIR2_EL2 = X[t, 64];
    else
        MAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR2_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR2_EL3, Extended Memory Attribute Indirection Register (EL3)

The MAIR2_EL3 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a VMSAv8-64 or VMSAv9-128 translation table entry for stage 1 translations at EL1.

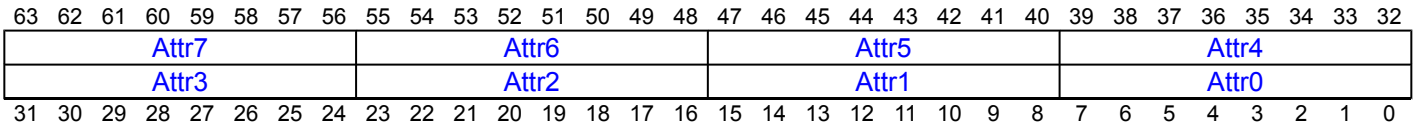
Configuration

This register is present only when FEAT_AIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR2_EL3 are UNDEFINED.

Attributes

MAIR2_EL3 is a 64-bit register.

Field descriptions



Attr<n>, bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 1, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 0, see MAIR_EL3.Attr

Attr is encoded as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of 'dd' for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise,UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise,UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR2_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MAIR2_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR2_EL3;
```

MSR MAIR2_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.MAIR2_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MAIR2_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR_EL1, Memory Attribute Indirection Register (EL1)

The MAIR_EL1 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL1.

Configuration

AArch64 System register MAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PRRR[31:0] when TTBCR.EAE == 0.

AArch64 System register MAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MAIR0[31:0] when TTBCR.EAE == 1.

AArch64 System register MAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register NMRR[31:0] when TTBCR.EAE == 0.

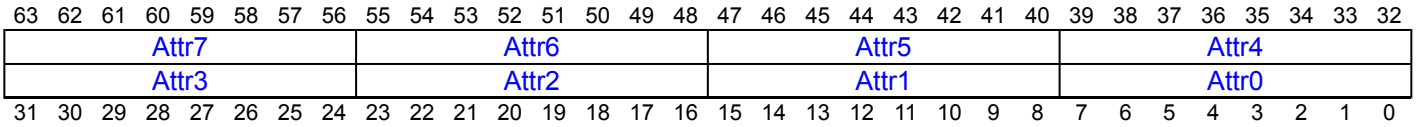
AArch64 System register MAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register MAIR1[31:0] when TTBCR.EAE == 1.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR_EL1 are UNDEFINED.

Attributes

MAIR_EL1 is a 64-bit register.

Field descriptions



MAIR_EL1 is permitted to be cached in a TLB.

Attr<n>, bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 0, or when FEAT_AIE is not implemented, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 1, see MAIR2_EL1.Attr

Attr is encoded as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of 'dd' for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise,UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise,UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name MAIR_EL1 or MAIR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MAIR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x140];
    else
        X[t, 64] = MAIR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = MAIR_EL2;
    else
        X[t, 64] = MAIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR_EL1;
```

```
MSR MAIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x140] = X[t, 64];
    else
        MAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        MAIR_EL2 = X[t, 64];
    else
        MAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, MAIR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x140];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = MAIR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = MAIR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR MAIR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x140] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        MAIR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        MAIR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR_EL2, Memory Attribute Indirection Register (EL2)

The MAIR_EL2 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL2.

Configuration

AArch64 System register MAIR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HMAIR0[31:0].

AArch64 System register MAIR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HMAIR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR_EL2 are UNDEFINED.

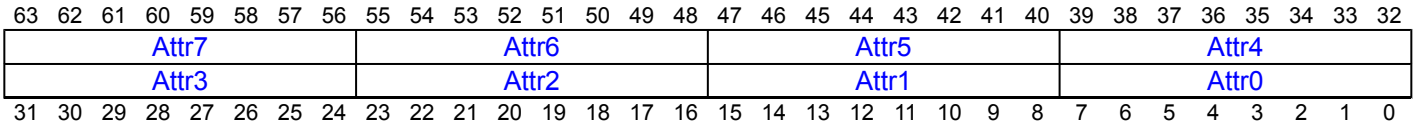
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MAIR_EL2 is a 64-bit register.

Field descriptions



MAIR_EL2 is permitted to be cached in a TLB.

Attr<n>, bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 0, or when FEAT_AIE is not implemented, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 1, see MAIR2_EL2.Attr

Attr is encoded as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of 'dd' for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise,UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise,UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name MAIR_EL2 or MAIR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MAIR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MAIR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR_EL2;
```

```
MSR MAIR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    MAIR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR_EL2 = X[t, 64];
```

```
MRS <Xt>, MAIR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x140];
    else
        X[t, 64] = MAIR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = MAIR_EL2;
    else
        X[t, 64] = MAIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR_EL1;
```

```
MSR MAIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x140] = X[t, 64];
    else
        MAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        MAIR_EL2 = X[t, 64];
    else
        MAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR_EL3, Memory Attribute Indirection Register (EL3)

The MAIR_EL3 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL3.

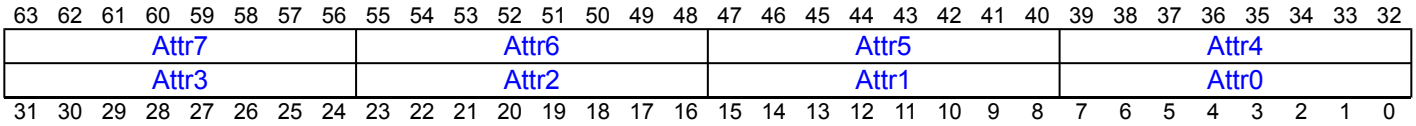
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR_EL3 are UNDEFINED.

Attributes

MAIR_EL3 is a 64-bit register.

Field descriptions



MAIR_EL3 is permitted to be cached in a TLB.

Attr<n>, bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 0, or when FEAT_AIE is not implemented, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 1, see MAIR2_EL3.Attr

Attr is encoded as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of 'dd' for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise,UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise,UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MAIR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR_EL3;
```

```
MSR MAIR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.MAIR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MAIR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDCCINT_EL1, Monitor DCC Interrupt Enable Register

The MDCCINT_EL1 characteristics are:

Purpose

Enables interrupt requests to be signaled based on the DCC status flags.

Configuration

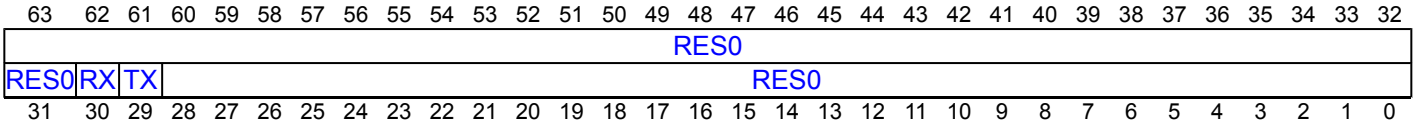
AArch64 System register MDCCINT_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDCCINT[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MDCCINT_EL1 are UNDEFINED.

Attributes

MDCCINT_EL1 is a 64-bit register.

Field descriptions



Bits [63:31]

Reserved, RES0.

RX, bit [30]

DCC interrupt request enable control for DTRRX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

RX	Meaning
0b0	No interrupt request generated by DTRRX.
0b1	Interrupt request will be generated on RXfull == 1.

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TX, bit [29]

DCC interrupt request enable control for DTRTX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

TX	Meaning
0b0	No interrupt request generated by DTRTX.
0b1	Interrupt request will be generated on TXfull == 0.

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [28:0]

Reserved, RES0.

Accessing MDCCINT_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDCCINT_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    X[t, 64] = MDCCINT_EL1;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCCINT_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCCINT_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDCCINT_EL1;
```

```
MSR MDCCINT_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    MDCCINT_EL1 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCCINT_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCCINT_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MDCCINT_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDCCSR_EL0, Monitor DCC Status Register

The MDCCSR_EL0 characteristics are:

Purpose

Read-only register containing control status flags for the DCC.

Configuration

AArch64 System register MDCCSR_EL0 bits [30:29] are architecturally mapped to External register EDSCR[30:29].

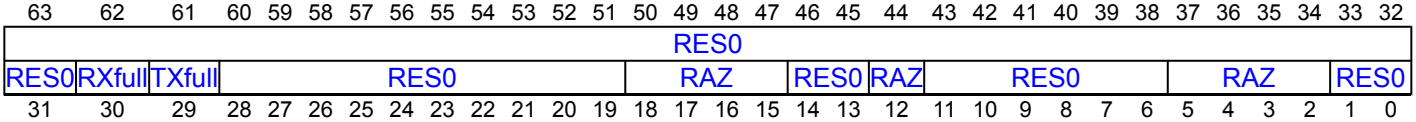
AArch64 System register MDCCSR_EL0 bits [30:29] are architecturally mapped to AArch32 System register DBGDSCRint[30:29].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MDCCSR_EL0 are UNDEFINED.

Attributes

MDCCSR_EL0 is a 64-bit register.

Field descriptions



Bits [63:31]

Reserved, RES0.

RXfull, bit [30]

DTRRX full. Read-only view of the equivalent bit in the EDSCR.

TXfull, bit [29]

DTRTX full. Read-only view of the equivalent bit in the EDSCR.

Bits [28:19]

Reserved, RES0.

Bits [18:15]

Reserved, RAZ.

Bits [14:13]

Reserved, RES0.

Bit [12]

Reserved, RAZ.

Bits [11:6]

Reserved, RES0.

Bits [5:2]

Reserved, RAZ.

Bits [1:0]

Reserved, RES0.

Accessing MDCCSR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDCCSR_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b0000	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    X[t, 64] = MDCCSR_EL0;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCCSR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCCSR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```

```
        X[t, 64] = MDCCSR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDCCSR_EL0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDCR_EL2, Monitor Debug Configuration Register (EL2)

The MDCR_EL2 characteristics are:

Purpose

Provides EL2 configuration options for self-hosted debug and the Performance Monitors Extension.

Configuration

AArch64 System register MDCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HDCR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MDCR_EL2 are UNDEFINED.

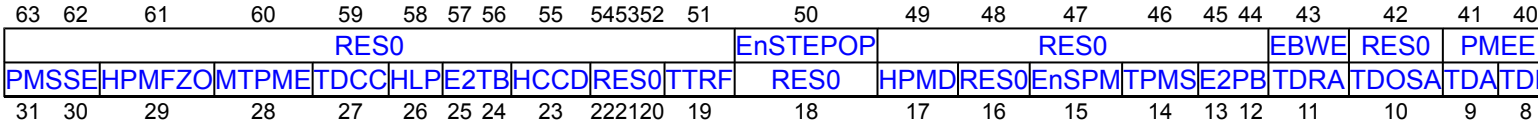
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MDCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:51]

Reserved, RES0.

EnSTEPOP, bit [50]

When FEAT_STEP2 is implemented:

EnSTEPOP	Meaning
0b0	Execution from MDSTEPOP_EL1 is disabled.
0b1	Execution from MDSTEPOP_EL1 is not disabled by this control.

If EL2 is not implemented or EL2 is disabled in the current Security state, then the Effective value of this field is 0b1, other than for a direct read of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [49:44]

Reserved, RES0.

EBWE, bit [43]

When FEAT_Debugv8p9 is implemented:

Extended Breakpoint and Watchpoint Enable. Enables use of additional breakpoints or watchpoints.

EBWE	Meaning
0b0	The Effective value of MDSCR_EL1.EMBWE is 0. The Effective value of MDSELR_EL1.BANK is zero at EL2.
0b1	The Effective values of MDSCR_EL1.EMBWE and MDSELR_EL1.BANK are not affected by this field.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

If EL2 is not implemented or EL2 is disabled in the current Security state, then the Effective value of this field is 1, other than for a direct read of the register.

This field is ignored by the PE and treated as 0 when EL3 is implemented and MDCR_EL3.EBWE is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

PMEE, bits [41:40]

When FEAT_EBEP is implemented:

Performance Monitors Exception Enable. Controls the generation of the **PMUIRQ** signal and the PMU Profiling exception at EL0, EL1, and EL2.

PMEE	Meaning
0b00	The **PMUIRQ** signal is asserted on a PMU overflow, and the PMU Profiling exception is disabled.
0b01	The **PMUIRQ** signal and the PMU Profiling exception are both controlled by PMECR_EL1.PMEE.
0b10	The **PMUIRQ** signal is deasserted, and the PMU Profiling exception is disabled.
0b11	The **PMUIRQ** signal is deasserted, and the PMU Profiling exception is enabled.

If EL2 is not implemented or EL2 is disabled in the current Security state, then the Effective value of this field is 0b01, other than for a direct read of the register.

This field is ignored by the PE when all of the following are true:

- EL3 is implemented.
- MDCR_EL3.PMEE != 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [39:37]

Reserved, RES0.

HPMFZS, bit [36]
When FEAT_SPEv1p2 is implemented:

Hyp Performance Monitors Freeze-on-SPE event. Stop counters when PMBLIMITR_EL1.{PMFZ,E} is {1,1} and profiling is stopped.

HPMFZS	Meaning
0b0	Do not freeze on a Statistical Profiling Buffer Management event.
0b1	Affected counters do not count following a Statistical Profiling Buffer Management event.

The pseudocode function `SPEProfilingStopped` describes when profiling is stopped.

The counters affected by this field are the event counters in the second range. This applies even when EL2 is disabled in the current Security state.

Other event counters and PMCCNTR_EL0 are not affected by this field.

When FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0 is not affected by this field.

If MDCR_EL2.HPMN is equal to `GetNumEventCountersSelfHosted()`, then this field has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:32]

Reserved, RES0.

PMSSE, bits [31:30]
When FEAT_PMUv3_SS is implemented:

Performance Monitors Snapshot Enable. Controls the generation of Capture events.

PMSSE	Meaning
0b00	Capture events are disabled.
0b01	Capture events are controlled by PMECR_EL1.SSE.
0b10	Capture events are enabled and prohibited.
0b11	Capture events are enabled and allowed.

If EL2 is not implemented, then the Effective value of this field is `0b01`.

The reset behavior of this field is:

- On a Cold reset:
  - When the highest implemented Exception level is EL2, this field resets to `'00'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPMFZO, bit [29]
When FEAT_PMUv3p7 is implemented:

Hyp Performance Monitors Freeze-on-overflow. Stop event counters on overflow.

HPMFZO	Meaning
0b0	Do not freeze on overflow.
0b1	Affected counters do not count when all of the following are true for any event counter PMEVCNTR<m>_EL0 in the second range: • PMOVSCLR_EL0[m] is 1. • Either FEAT_SEBEP is not implemented or PMEVTYPER<m>_EL0.SYNC is 0.

The counters affected by this field are the event counters in the second range. This applies even when EL2 is disabled in the current Security state.

Other event counters and PMCCNTR_EL0 are not affected by this field.

When FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0 is not affected by this field.

If MDCR_EL2.HPMN is equal to `GetNumEventCountersSelfHosted()`, then this field has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTPME, bit [28]
When FEAT_MTPMU is implemented and EL3 is not implemented:

Multi-threaded PMU Enable. Enables use of the PMEVTYPER<n>_EL0.MT bits.

MTPME	Meaning
0b0	FEAT_MTPMU is disabled. The Effective value of PMEVTYPER<n>_EL0.MT is 0.
0b1	PMEVTYPER<n>_EL0.MT bits not affected by this field.

If FEAT_MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this field is 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'1'`.

Otherwise:

Reserved, RES0.

TDCC, bit [27]
When FEAT_FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel at EL1 and EL0 to EL2.

TDCC	Meaning
0b0	This control does not cause any register accesses to be trapped.
0b1	If EL2 is implemented and enabled in the current Security state, accesses to the DCC registers at EL1 and EL0 generate a Trap exception to EL2, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state.

The DCC registers trapped by this control are:

AArch64: OSDTRRX_EL1, OSDTRTX_EL1, MDCCSR_EL0, MDCCINT_EL1, and, when the PE is in Non-debug state, DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0.

AArch32: DBGDTRRXext, DBGDTRTXext, DBGDSCRint, DBGDCCINT, and, when the PE is in Non-debug state, DBGDTRRXint and DBGDTRTXint.

The traps are reported with EC syndrome value:

- 0x05 for trapped AArch32 MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped AArch32 LDC to DBGDTRTXint and STC from DBGDTRRXint.
- 0x18 for trapped AArch64 MRS and MSR accesses.

When the PE is in Debug state, MDCR_EL2.TDCC does not trap any accesses to:

AArch64: DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0.

AArch32: DBGDTRRXint and DBGDTRTXint.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HLP, bit [26]

When FEAT_PMUv3p5 is implemented:

Hypervisor Long Event Counter Enable. Determines which event counter bit generates an overflow recorded by PMOVSR[n].

HLP	Meaning
0b0	Affected counters overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[31:0].
0b1	Affected counters overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[63:0].

When FEAT_EBEP is implemented and the PMU Profiling exception is enabled, the Effective value of this field is 1.

The counters affected by this field are the event counters in the second range. This applies even when EL2 is disabled in the current Security state.

The following are not affected by this field:

- Other event counters.
- PMCCNTR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0.

If MDCR_EL2.HPMN is equal to GetNumEventCountersSelfHosted(), then this field has no effect.

For more information see the description of MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E2TB, bits [25:24]
When FEAT_TRBE is implemented:

EL2 Trace Buffer.

If EL2 is implemented and enabled in the trace buffer owning Security state, then this field controls the trace buffer owning translation regime.

If EL2 is implemented and enabled in the current Security state, then this field controls access to trace buffer System registers from EL1.

E2TB	Meaning
0b00	If EL2 is implemented and enabled in the trace buffer owning Security state, then the trace buffer owning Exception level is EL2. Otherwise, the trace buffer owning Exception level is EL1 and, if TraceBufferEnabled() == TRUE, tracing is prohibited at EL2. If EL2 is implemented and enabled in the current Security state, then accesses to trace buffer System registers at EL1 are trapped to EL2, unless the access generates a higher priority exception.
0b10	Trace buffer owning Exception level is EL1. If TraceBufferEnabled() == TRUE, then tracing is prohibited at EL2. If EL2 is implemented and enabled in the current Security state, then accesses to trace buffer System registers at EL1 are trapped to EL2, unless the access generates a higher priority exception.
0b11	Trace buffer owning Exception level is EL1. If TraceBufferEnabled() == TRUE, then tracing is prohibited at EL2. Accesses to trace buffer System registers at EL1 are not trapped by this mechanism.

All other values are reserved.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to TRBBASER_EL1, TRBLIMITR_EL1, TRBMAR_EL1, TRBPTR_EL1, TRBSR_EL1, and TRBTRG_EL1.
- If FEAT_TRBE_MPAM is implemented, MRS and MSR accesses to TRBMPAM_EL1.
- If FEAT_TRBE_EXC is implemented, MRS and MSR accesses to TRBSR_EL2 and TRBSR_EL12.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL2, reported using EC syndrome value 0x18.

When TraceBufferEnabled()==FALSE, this field has no effect on whether tracing is prohibited.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HCCD, bit [23]
When FEAT_PMUv3p5 is implemented:

Hypervisor Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting at EL2.

HCCD	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited at EL2.

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

Bits [22:20]

Reserved, RES0.

TTRF, bit [19]

When FEAT_TRF is implemented:

Traps use of the Trace Filter Control registers at EL1 to EL2, as follows:

- Access to TRFCR_EL1 is trapped to EL2, reported using EC syndrome value `0x18`.
- Access to TRFCR is trapped to EL2, reported using EC syndrome value `0x03`.

TTRF	Meaning
`0b0`	Accesses to the specified registers at EL1 are not affected by this control.
`0b1`	Accesses to the specified registers at EL1 generate a trap exception to EL2 when EL2 is enabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

HPMD, bit [17]

When FEAT_PMUv3p1 is implemented and FEAT_Debugv8p2 is implemented:

Guest Performance Monitors Disable. Controls PMU operation at EL2.

HPMD	Meaning
`0b0`	Counters are not affected by this mechanism.
`0b1`	Affected counters are prohibited from counting at EL2. If PMCR_EL0.DP is 1, then PMCCNTR_EL0 is disabled at EL2. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.

The counters affected by this field are:

- Event counters in the first range.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- If PMCR_EL0.DP is 1, the cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When FEAT_PMUv3p1 is implemented:

Guest Performance Monitors Disable. Controls PMU operation at EL2 when ExternalSecureNoninvasiveDebugEnabled() is FALSE.

HPMD	Meaning
0b0	Counters are not affected by this mechanism.
0b1	If ExternalSecureNoninvasiveDebugEnabled() is FALSE then all the following apply: • Affected event counters are prohibited from counting at EL2. • If PMCR_EL0.DP is 1, then PMCCNTR_EL0 is disabled at EL2. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.

If ExternalSecureNoninvasiveDebugEnabled() is TRUE then the event counters and PMCCNTR_EL0 are not affected by this field.

Otherwise, the counters affected by this field are:

- Event counters in the first range.
- If PMCR_EL0.DP is 1, the cycle counter, PMCCNTR_EL0.

Other event counters are not affected by this field. When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [16]

Reserved, RES0.

EnSPM, bit [15]

When FEAT_SPMU is implemented:

Enable access to System PMU registers. When disabled, accesses to System PMU registers generate a trap to EL2.

EnSPM	Meaning
0b0	Accesses of the specified System PMU registers at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified System PMU registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to SPMACCESSR_EL1, SPMCFGR_EL1, SPMCGCR<n>_EL1, SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMDEVAFF_EL1, SPMDEVARCH_EL1, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMIIDR_EL1, SPMINTENCLR_EL1, SPMINTENSET_EL1, SPMOVSCLR_EL0, SPMOVSSET_EL0, SPMSCR_EL1, and SPMSELR_EL0.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPMS, bit [14]

When FEAT_SPE is implemented:

Trap Performance Monitor Sampling. Enables a trap to EL2 on accesses of SPE registers.

TPMS	Meaning
0b0	Accesses of the specified SPE registers are not trapped by this mechanism.
0b1	Accesses of the specified SPE registers at EL1 are trapped to EL2, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to PMSCR_EL1, PMSEVFR_EL1, PMSFCR_EL1, PMSICR_EL1, PMSIRR_EL1, and PMSLATFR_EL1.
- MRS accesses to PMSIDR_EL1.
- If FEAT_SPE_FnE is implemented, MRS and MSR accesses to PMSNEVFR_EL1.
- If FEAT_SPE_FDS is implemented, MRS and MSR accesses to PMSDSFR_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E2PB, bits [13:12]

When FEAT_SPE is implemented:

EL2 Profiling Buffer.

If EL2 is implemented and enabled in the Profiling Buffer owning Security state, then this field controls the Profiling Buffer owning translation regime.

If EL2 is implemented and enabled in the current Security state, then this field controls access to Profiling Buffer System registers from EL1.

E2PB	Meaning
0b00	If EL2 is implemented and enabled in the Profiling Buffer owning Security state, then the Profiling Buffer owning Exception level is EL2. Otherwise, the Profiling Buffer owning Exception level is EL1 and, Profiling is prohibited at EL2. If EL2 is implemented and enabled in the current Security state, then accesses to Profiling Buffer System registers at EL1 are trapped to EL2, unless the access generates a higher priority exception.
0b10	Profiling Buffer owning Exception level is EL1. Profiling is prohibited at EL2. If EL2 is implemented and enabled in the current Security state, then accesses to Profiling Buffer System registers at EL1 are trapped to EL2, unless the access generates a higher priority exception.
0b11	Profiling Buffer owning Exception level is EL1. Profiling is prohibited at EL2. Accesses to Profiling Buffer System registers at EL1 are not trapped by this mechanism.

All other values are reserved.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to PMBLIMITR_EL1, PMBPTR_EL1, and PMBSR_EL1.
- If FEAT_SPE_nVM is implemented, MRS and MSR accesses to PMBMAR_EL1.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL2, reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TDRA, bit [11]

Trap Debug ROM Address register access. Traps System register accesses to the Debug ROM registers to EL2 when EL2 is enabled in the current Security state as follows:

- If EL1 is using AArch64, accesses to MDRAR_EL1 are trapped to EL2, reported using EC syndrome value 0x18.
- If EL0 or EL1 is using AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x05 and MRRC or MCRR accesses are trapped to EL2, reported using EC syndrome value 0x0C:
  - DBGDRAR, DBGDSAR.

TDRA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0 and EL1 System register accesses to the Debug ROM registers are trapped to EL2 when EL2 is enabled in the current Security state, unless it is trapped by the following: • DBGDSCRext.UDCCdis. • MDSCR_EL1.TDCC.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

Note

EL2 does not provide traps on debug register accesses through the optional memory-mapped external debug interfaces.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDOSA, bit [10]

When FEAT_DoubleLock is implemented:

Trap debug OS-related register access. Traps EL1 System register accesses to the powerdown debug registers to EL2, from both Execution states as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
  - OSLAR_EL1, OSLSR_EL1, OSDLR_EL1, and DBGPRCR_EL1.
  - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x05:
  - DBGOSLSR, DBGOSLAR, DBGOSDLR, and DBGPRCR.
  - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

TDOSA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 System register accesses to the powerdown debug registers are trapped to EL2 when EL2 is enabled in the current Security state.

Note

These registers are not accessible at EL0.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trap debug OS-related register access. Traps EL1 System register accesses to the powerdown debug registers to EL2, from both Execution states as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x18`:
  - OSLAR_EL1, OSLSR_EL1, and DBGPRCR_EL1.
  - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x05`:
  - DBGOSLSR, DBGOSLAR, and DBGPRCR.
  - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

It is IMPLEMENTATION DEFINED whether accesses to OSDLR_EL1 are trapped.

It is IMPLEMENTATION DEFINED whether accesses to DBGOSDLR are trapped.

TDOSA	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL1 System register accesses to the powerdown debug registers are trapped to EL2 when EL2 is enabled in the current Security state.

Note

These registers are not accessible at EL0.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

Note

EL2 does not provide traps on debug register accesses through the optional memory-mapped external debug interfaces.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDA, bit [9]

Trap accesses of debug System registers. Enables a trap to EL2 on accesses of debug System registers.

TDA	Meaning
`0b0`	Accesses of the specified debug System registers are not trapped by this mechanism.
`0b1`	Accesses of the specified debug System registers at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to DBGAUTHSTATUS_EL1, DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGCLAIMCLR_EL1, DBGCLAIMSET_EL1, DBGWCR<n>_EL1, DBGWVR<n>_EL1, MDCCINT_EL1, MDCCSR_EL0, MDSCR_EL1, OSDTRRX_EL1, OSDTRTX_EL1, and OSECCR_EL1.
- If FEAT_Debugv8p9 is implemented, MRS and MSR accesses to MDSELR_EL1.
- If FEAT_STEP2 is implemented, MRS and MSR accesses to MDSTEPOP_EL1.
- In Non-debug state, MRS accesses to DBGDTRRX_EL0 and DBGDTR_EL0 and MSR accesses to DBGDTRTX_EL0 and DBGDTR_EL0.

In AArch32 state, the instructions affected by this control are:

- MRC and MCR accesses to DBGAUTHSTATUS, DBGBCR<n>, DBGBVR<n>, DBGBXVR<n>, DBGCLAIMCLR, DBGCLAIMSET, DBGDCCINT, DBGDEVID, DBGDEVID1, DBGDEVID2, DBGDIDR, DBGDSCRext, DBGDSCRint, DBGDTRRXext, DBGDTRTXext, DBGOSECCR, DBGVCR, DBGWCR<n>, DBGWFAR, and DBGWVR<n>.
- STC accesses to DBGDTRRXint and LDC accesses to DBGDTRTXint.
- In Non-debug state, MRC accesses to DBGDTRRXint and MCR accesses to DBGDTRTXint.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Trapped AArch32 instructions are reported using EC syndrome value 0x05 for MRC and MCR accesses, and 0x06 for LDC and STC accesses.

The following instructions are not trapped in Debug state:

- AArch64 MRS accesses to DBGDTRRX_EL0 and DBGDTR_EL0 and MSR accesses to DBGDTRTX_EL0 and DBGDTR_EL0.
- AArch32 MRC accesses to DBGDTRRXint and MCR accesses to DBGDTRTXint.

If 16 or fewer breakpoints and 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI, then it is IMPLEMENTATION DEFINED whether AArch64 accesses to MDSELR_EL1 are trapped to EL2 when MDCR_EL2.TDA is 1.

This field is ignored by the PE and treated as one when any of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDE, bit [8]

Trap Debug Exceptions. Controls routing of Debug exceptions, and defines the debug target Exception level, EL_D.

TDE	Meaning
0b0	The debug target Exception level is EL1.
0b1	If EL2 is enabled for the current Effective value of SCR_EL3.NS, the debug target Exception level is EL2, otherwise the debug target Exception level is EL1. The MDCR_EL2.{TDRA, TDOSA, TDA} fields are treated as being 1 for all purposes other than returning the result of a direct read of the register.

For more information, see 'Routing debug exceptions'.

This field is treated as being 1 for all purposes other than a direct read when HCR_EL2.TGE == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HPME, bit [7]

When FEAT_PMUv3 is implemented:

Hyp Enable.

HPME	Meaning
0b0	Affected counters are disabled and do not count.
0b1	Affected counters are enabled by PMCNTENSET_EL0.

The counters affected by this field are the event counters in the second range. This applies even when EL2 is disabled in the current Security state.

The following counters are not affected by this field:

- Other event counters.
- PMCCNTR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0.

If MDCR_EL2.HPMN is equal to `GetNumEventCountersSelfHosted()`, then this field has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPM, bit [6]

When FEAT_PMUv3 is implemented:

Trap accesses of PMU registers. Enables a trap to EL2 on accesses of PMU registers.

TPM	Meaning
0b0	Accesses of the specified PMU registers are not trapped by this mechanism.
0b1	Accesses of the specified PMU registers at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to PMCCFILTR_EL0, PMCCNTR_EL0, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMINTENCLR_EL1, PMINTENSET_EL1, PMOVSCLR_EL0, PMOVSSET_EL0, PMSELR_EL0, PMSWINC_EL0, PMUSERENR_EL0, PMXEVCNTR_EL0, and PMXEVTYPER_EL0.
- MRS accesses to PMCEID0_EL0 and PMCEID1_EL0.
- If FEAT_PMUv3p4 is implemented, MRS accesses to PMMIR_EL1.
- If FEAT_PMUv3p9 is implemented, MSR accesses to PMZR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, MRS accesses to PMICFILTR_EL0 and PMICNTR_EL0.
- If FEAT_EBEP is implemented or FEAT_PMUv3_SS is implemented, MRS and MSR accesses to PMECR_EL1.
- If FEAT_SEBEP is implemented, MRS and MSR accesses to PMIAR_EL1.

In AArch32 state, the instructions affected by this control are:

- MRC and MCR accesses to PMCCFILTR, PMCCNTR, PMCNTENCLR, PMCNTENSET, PMCR, PMEVCNTR<n>, PMEVTYPER<n>, PMINTENCLR, PMINTENSET, PMOVSR, PMOVSSET, PMSELR, PMSWINC, PMUSERENR, PMXEVCNTR, and PMXEVTYPER.
- MRC accesses to PMCEID0 and PMCEID1.
- MRRC and MCRR accesses to PMCCNTR.
- If FEAT_PMUv3p1 is implemented, MRC accesses to PMCEID2 and PMCEID3.
- If FEAT_PMUv3p4 is implemented, MRC accesses to PMMIR.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Trapped AArch32 instructions are reported using EC syndrome value 0x03 for MRC and MCR accesses, and 0x04 for MRRC and MCRR accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPMCR, bit [5]

When FEAT_PMUv3 is implemented:

Trap PMCR_EL0 or PMCR accesses. Traps EL0 and EL1 accesses to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to PMCR_EL0 are trapped to EL2, reported using EC syndrome value `0x18`.
- In AArch32 state, accesses to PMCR are trapped to EL2, reported using EC syndrome value `0x03`.

TPMCR	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 and EL1 accesses to the specified registers are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by the following: • PMUSERENR.EN. • PMUSERENR_EL0.EN.

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPMN, bits [4:0]

When FEAT_PMUv3 is implemented:

Defines the number of event counters PMEVCNTR<n>_EL0 and, if FEAT_PMUv3_SS is implemented, event counter snapshot registers PMEVCNTSVR<n>_EL1, that are accessible from EL1 and, if permitted, from EL0.

MDCR_EL2.HPMN divides the event counters accessible from self-hosted software into a first range and a second range.

When FEAT_PMUv3_EXTPMN is implemented, all of the following apply:

- PMCCR.EPMN divides the NUM_PMU_COUNTERS event counters implemented by the PE into the event counters that MDCR_EL2.HPMN divides into the first and second ranges, and a third range that is inaccessible from self-hosted software.
- If MDCR_EL2.HPMN is not 0 and is less than the Effective value of PMCCR.EPMN, then event counters [0..(MDCR_EL2.HPMN-1)] are in the first range, and the remaining event counters [MDCR_EL2.HPMN..(PMCCR.EPMN-1)] are in the second range.
- The pseudcode function `GetNumEventCountersSelfHosted()` returns the Effective value of PMCCR.EPMN.

When FEAT_PMUv3_EXTPMN is not implemented, all of the following apply:

- All of the NUM_PMU_COUNTERS event counters are accessible to self-hosted software and no counters are in the third range.
- If MDCR_EL2.HPMN is not 0 and is less than NUM_PMU_COUNTERS, then event counters [0..(MDCR_EL2.HPMN-1)] are in the first range, and the remaining event counters [MDCR_EL2.HPMN..(NUM_PMU_COUNTERS-1)] are in the second range.
- The pseudocode function `GetNumEventCountersSelfHosted()` returns NUM_PMU_COUNTERS.

If FEAT_HPMN0 is implemented and MDCR_EL2.HPMN is 0, then all of the following apply:

- No event counters are in the first range.
- If FEAT_PMUv3_EXTPMN is implemented, then event counters [0..(PMCCR.EPMN-1)] are in the second range.
- If FEAT_PMUv3_EXTPMN is not implemented, then all event counters are in the second range.

If MDCR_EL2.HPMN is equal to `GetNumEventCountersSelfHosted()`, or EL2 is not implemented, then all of the following apply:

- If FEAT_PMUv3_EXTPMN is implemented, then event counters [0..(PMCCR.EPMN-1)] are in the first range.
- If FEAT_PMUv3_EXTPMN is not implemented, then all event counters are in the first range.
- No event counters are in the second range.

All of the following apply for an event counter PMEVCNTR<n>_EL0 in the first range:

- The counter is accessible from EL1, EL2, and EL3.
- The counter is accessible from EL0 if permitted by PMUSERENR_EL0 and PMUACR_EL1, or by PMUSERENR.
- If FEAT_PMUv3p5 is implemented, then PMCR_EL0.LP or PMCR.LP determines whether the counter overflow flag PMOVSSET_EL0[n] is set on unsigned overflow of PMEVCNTR<n>_EL0[31:0] or PMEVCNTR<n>_EL0[63:0].
- PMCR_EL0.E and PMCNTENSET_EL0[n] enable the operation of the event counter.

All of the following apply for an event counter PMEVCNTR<n>_EL0 in the second range:

- The counter is accessible from EL2 and EL3.
- If EL2 is disabled in the current Security state, then the event counter is accessible from EL1, and from EL0 if permitted by PMUSERENR_EL0 and PMUACR_EL1, or by PMUSERENR.
- If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP determines whether the counter overflow flag PMOVSSET_EL0[n] is set on unsigned overflow of PMEVCNTR<n>_EL0[31:0] or PMEVCNTR<n>_EL0[63:0].
- MDCR_EL2.HPME and PMCNTENSET_EL0[n] enable the operation of the event counter.

If FEAT_PMUv3_SS is implemented, then all of the following apply:

- For an event counter snapshot register PMEVCNTSVR<n>_EL1 in the first range, the register is accessible from EL1, EL2, and EL3.
- For an event counter snapshot register PMEVCNTSVR<n>_EL1 in the second range, the register is accessible from EL2 and EL3. If EL2 is disabled in the current Security state, the event counter is also accessible from EL1.

For information about counters in the third range, see the description of PMCCR.EPMN.

Values greater than `GetNumEventCountersSelfHosted()` are reserved. If FEAT_HPMN0 is not implemented then the value 0 is reserved.

When FEAT_PMUv3_EXTPMN is not implemented and this field is set to a reserved value, the following CONSTRAINED UNPREDICTABLE behaviors apply:

- The value returned by a direct read of MDCR_EL2.HPMN is UNKNOWN.
- The number of event counters in each of the first and second ranges is UNKNOWN. That is, either:
  - The PE behaves as if MDCR_EL2.HPMN is set to an UNKNOWN nonzero value less than or equal to NUM_PMU_COUNTERS.
  - All counters are in the second range and none are in the first range.

When FEAT_PMUv3_EXTPMN is implemented and this field is set to a reserved value, the following behaviors apply:

- The value returned by a direct read of MDCR_EL2.HPMN is the Effective value of PMCCR.EPMN.
- No event counters are in the second range.
- The value returned by an indirect read of MDCR_EL2.HPMN as a result of direct reads of PMCR_EL0.N or PMCR.N is the Effective value of PMCCR.EPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to the expression `NUM_PMU_COUNTERS`.

Otherwise:

Reserved, RES0.

Accessing MDCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDCR_EL2;
```

MSR MDCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MDCR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDCR_EL3, Monitor Debug Configuration Register (EL3)

The MDCR_EL3 characteristics are:

Purpose

Provides EL3 configuration options for self-hosted debug and the Performance Monitors Extension.

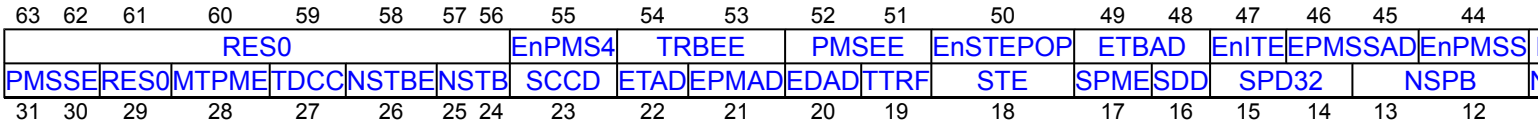
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MDCR_EL3 are UNDEFINED.

Attributes

MDCR_EL3 is a 64-bit register.

Field descriptions



Bits [63:56]

Reserved, RES0.

EnPMS4, bit [55]

When FEAT_SPE_nVM is implemented:

Enable access to SPE registers. When disabled, accesses to SPE registers generate a trap to EL3.

EnPMS4	Meaning
0b0	Accesses of the specified SPE registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified SPE registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to PMBMAR_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBEE, bits [54:53]

When FEAT_TRBE_EXC is implemented:

Trace buffer management event Exception Enable.

TRBEE	Meaning
0b00	Disabled. TRBE Profiling exceptions for all Exception levels are disabled. All of the following apply: • No trace buffer management events are recorded in TRBSR_EL3. • TRBE Profiling exceptions are not generated. • TRBSR_EL1.IRQ drives the interrupt request signal **TRBIRQ**. • Accesses to TRBSR_EL2 at EL2 are trapped to EL3. • Accesses to TRBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1 and accesses to TRBSR_EL1 at EL2 ignore the value of HCR_EL2.E2H.
0b01	Delegated. TRBE Profiling exceptions for EL3 are disabled, but might be enabled for EL2 or EL1 by TRFCR_EL2.EE or TRFCR_EL1.EE. All of the following apply: • No trace buffer management events are recorded in TRBSR_EL3. • TRBSR_EL3.IRQ is ignored and TRBE Profiling exceptions are not taken to EL3.
0b10	Enabled. TRBE Profiling exceptions for EL3 are enabled for trace buffer management events targeting EL3, as follows: • Trace buffer management events due to a fault on a write to the trace buffer that would generate a Data Abort exception taken to EL3 if generated by a store instruction executed at the owning Exception level are recorded in TRBSR_EL3. That is, any of the following faults: ◦ Granule Protection Check faults other than Granule Protection Faults (GPFs). ◦ If SCR_EL3.GPF is 1, GPFs. ◦ If SCR_EL3.EA is 1, External aborts. • TRBE Profiling exceptions are generated and taken to EL3 when unmasked and TRBSR_EL3.IRQ is 1.
0b11	Trap all. TRBE Profiling exceptions for EL3 are enabled for all trace buffer management events, as follows: • All trace buffer management events are recorded in TRBSR_EL3. • TRBE Profiling exceptions are generated and taken to EL3 when unmasked and TRBSR_EL3.IRQ is 1.

If EL3 is not implemented, then the Effective value of MDCR_EL3.TRBEE is 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSEE, bits [52:51]

When FEAT_SPE_EXC is implemented:

Profiling Buffer management event Exception Enable.

PMSEE	Meaning
0b00	Disabled. SPE Profiling exceptions for all Exception levels are disabled. All of the following apply: • No Profiling Buffer management events are recorded in PMBSR_EL3. • SPE Profiling exceptions are not generated. • PMBSR_EL1.S drives the interrupt request signal **PMBIRQ**. • Accesses to PMBSR_EL2 at EL2 are trapped to EL3. • Accesses to PMBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1 and accesses to PMBSR_EL1 at EL2 ignore the value of HCR_EL2.E2H.
0b01	Delegated. SPE Profiling exceptions for EL3 are disabled, but might be enabled for EL2 or EL1 by PMSCR_EL2.EE or PMSCR_EL1.EE. All of the following apply: • No Profiling Buffer management events are recorded in PMBSR_EL3. • PMBSR_EL3.S is ignored and SPE Profiling exceptions are not taken to EL3.
0b10	Enabled. SPE Profiling exceptions for EL3 are enabled for Profiling Buffer management events targeting EL3, as follows: • Profiling Buffer management events due to a fault on a write to the Profiling Buffer that would generate a Data Abort exception taken to EL3 if generated by a store instruction executed at the owning Exception level are recorded in PMBSR_EL3. That is, any of the following faults: ◦ Granule Protection Check faults other than Granule Protection Faults (GPFs). ◦ If SCR_EL3.GPF is 1, GPFs. ◦ If SCR_EL3.EA is 1, External aborts. • SPE Profiling exceptions are generated and taken to EL3 when unmasked and PMBSR_EL3.S is 1.
0b11	Trap all. SPE Profiling exceptions for EL3 are enabled for all Profiling Buffer management events, as follows: • All Profiling Buffer management events are recorded in PMBSR_EL3. • SPE Profiling exceptions are generated and taken to EL3 when unmasked and PMBSR_EL3.S is 1.

If EL3 is not implemented, then the Effective value of MDCR_EL3.PMSEE is 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSTEPOP, bit [50]

When FEAT_STEP2 is implemented:

EnSTEPOP	Meaning
0b0	Execution from MDSTEPOP_EL1 is disabled. EL2, EL1 and EL0 System register accesses to MDSTEPOP_EL1 are trapped to EL3, unless the access generates a higher priority exception.
0b1	Execution from MDSTEPOP_EL1 is not disabled by this control. System register accesses to MDSTEPOP_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ETBAD, bits [49:48]

When FEAT_TRBE_EXT is implemented:

External Trace Buffer Access Disable. Controls access to the Trace Buffer registers from an external debugger.

ETBAD	Meaning	Applies when
0b00	Non-secure accesses from an external debugger to Trace Buffer registers are prohibited. If FEAT_RME is implemented, Secure and Realm accesses from an external debugger to Trace Buffer registers are prohibited and Root accesses to Trace Buffer registers are allowed. If FEAT_RME is not implemented, Secure accesses to Trace Buffer registers are allowed.	
0b01	Secure and Non-secure accesses from an external debugger to Trace Buffer registers are prohibited. Root and Realm accesses to Trace Buffer registers are allowed.	When FEAT_RME is implemented
0b10	Realm and Non-secure accesses from an external debugger to Trace Buffer registers are prohibited. Root and Secure accesses to Trace Buffer registers are allowed.	When FEAT_RME is implemented
0b11	All accesses from an external debugger to Trace Buffer registers are allowed.	

If EL3 is not implemented, then the Effective value of this field is 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

Otherwise:

Reserved, RES0.

EnITE, bit [47]

When FEAT_ITE is implemented:

Enable access to Instrumentation trace registers. When disabled, accesses to Instrumentation trace registers generate a trap to EL3.

EnITE	Meaning
0b0	Accesses of the specified Instrumentation trace registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified Instrumentation trace registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to TRCITECR_EL1, TRCITECR_EL2, and TRCITECR_EL12.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EPMSSAD, bits [46:45]

When FEAT_PMUv3_SS is implemented:

External PMU Snapshot Access Disable. Controls access to the PMU Snapshot registers from an external debugger.

External accesses of the following registers are affected by this control:

- PMCCNTSVR_EL1, PMEVCNTSVR<n>_EL1, and PMSSCR_EL1.
- If FEAT_PMUv3_ICNTR is implemented, PMICNTSVR_EL1.

EPMSSAD	Meaning	Applies when
0b00	Non-secure accesses from an external debugger to the affected PMU Snapshot registers are prohibited. If FEAT_RME is implemented, Secure and Realm accesses from an external debugger to the affected PMU Snapshot registers are prohibited. Other accesses from an external debugger to the specified registers are not affected by this control.	
0b01	Secure and Non-secure accesses from an external debugger to the affected PMU Snapshot registers are prohibited. Other accesses from an external debugger to the specified registers are not affected by this control.	When FEAT_RME is implemented
0b10	Realm and Non-secure accesses from an external debugger to the affected PMU Snapshot registers are prohibited. Other accesses from an external debugger to the specified registers are not affected by this control.	When FEAT_RME is implemented
0b11	No accesses from an external debugger to the affected PMU Snapshot registers are prohibited by this control.	

If EL3 is not implemented, then the Effective value of this field is 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

Otherwise:

Reserved, RES0.

EnPMSS, bit [44]

When FEAT_PMUv3_SS is implemented:

Enable access to PMU Snapshot registers. When disabled, accesses to PMU Snapshot registers generate a trap to EL3.

EnPMSS	Meaning
0b0	Accesses of the specified PMU Snapshot registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified PMU Snapshot registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to PMCCNTSVR_EL1, PMEVCNTSVR<n>_EL1, and PMSSCR_EL1.
- If FEAT_PMUv3_ICNTR is implemented, MRS and MSR accesses to PMICNTSVR_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EBWE, bit [43]
When FEAT_Debugv8p9 is implemented:

Extended Breakpoint and Watchpoint Enable. Enables use of additional breakpoints or watchpoints, and enables a trap to EL3 on accesses to debug registers.

EBWE	Meaning
0b0	The Effective values of MDSCR_EL1.EMBWE and MDCR_EL2.EBWE are 0. The Effective value of MDSELR_EL1.BANK is zero at EL3. Accesses of MDSELR_EL1 at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	The Effective values of MDSCR_EL1.EMBWE, MDCR_EL2.EBWE, and MDSELR_EL1.BANK are not affected by this field. Accesses of MDSELR_EL1 are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to MDSELR_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

If EL3 is not implemented, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

EnPMS3, bit [42]
When FEAT_SPE_FDS is implemented:

Enable access to SPE registers. When disabled, accesses to SPE registers generate a trap to EL3.

EnPMS3	Meaning
0b0	Accesses of the specified SPE registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified SPE registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to PMSDSFR_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMEE, bits [41:40]

When FEAT_EBEP is implemented:

Performance Monitors Exception Enable. Controls the generation of the **PMUIRQ** signal and the PMU Profiling exception at all Exception levels.

PMEE	Meaning
0b00	The **PMUIRQ** signal is asserted on a PMU overflow, and the PMU Profiling exception is disabled.
0b01	The **PMUIRQ** signal and the PMU Profiling exception are both controlled by MDCR_EL2.PMEE.
0b10	The **PMUIRQ** signal is deasserted, and the PMU Profiling exception is disabled.
0b11	The **PMUIRQ** signal is deasserted, and the PMU Profiling exception is enabled.

If EL3 is not implemented, then the Effective value of this field is 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTB2, bit [39]

When FEAT_TRBE_MPAM is implemented:

Enable access to Trace Buffer registers. When disabled, accesses to Trace Buffer registers generate a trap to EL3.

EnTB2	Meaning
0b0	Accesses of the specified Trace Buffer registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified Trace Buffer registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to TRBMPAM_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E3BREC, bit [38]

When FEAT_BRBEv1p1 is implemented:

Branch Record Buffer EL3 Cold Reset Enable. With MDCR_EL3.E3BREW, controls branch recording at EL3.

E3BREC	Meaning
0b0	When MDCR_EL3.E3BREW == 0: Branch recording at EL3 is disabled. When MDCR_EL3.E3BREW == 1: Branch recording at EL3 is enabled.
0b1	When MDCR_EL3.E3BREW == 0: Branch recording at EL3 is enabled. When MDCR_EL3.E3BREW == 1: Branch recording at EL3 is disabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

E3BREW, bit [37]
When FEAT_BRBEv1p1 is implemented:

Branch Record Buffer EL3 Warm Reset Enable. With MDCR_EL3.E3BREC, controls branch recording at EL3.

For a description of the values derived by evaluating MDCR_EL3.E3BREC and MDCR_EL3.E3BREW together, see MDCR_EL3.E3BREC.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

EnPMSN, bit [36]
When FEAT_SPE_FnE is implemented:

Trap accesses to PMSNEVFR_EL1. Controls access to Statistical Profiling PMSNEVFR_EL1 System register from EL2 and EL1.

EnPMSN	Meaning
`0b0`	Accesses to PMSNEVFR_EL1 at EL2 and EL1 generate a Trap exception to EL3.
`0b1`	Do not trap PMSNEVFR_EL1 to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MPMX, bit [35]
When FEAT_PMUv3p7 is implemented:

Monitor Performance Monitors Extended control. With MDCR_EL3.SPME, controls PMU operation at EL3.

MPMX	Meaning
`0b0`	Counters are not affected by this mechanism.
`0b1`	Affected counters are prohibited from counting at EL3. If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled at EL3. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.

The counters affected by this field are:

- If EL2 is implemented and MDCR_EL3.SPME is 1, event counters PMEVCNTR<n>_EL0 in the first range.
- If EL2 is not implemented or MDCR_EL3.SPME is 0, event counters in the first and second ranges.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter, PMICNTR_EL0.
- If PMCR_EL0.DP is 1, the cycle counter, PMCCNTR_EL0.

Other event counters are not affected by this field. When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

MCCD, bit [34]
When FEAT_PMUv3p7 is implemented:

Monitor Cycle Counter Disable. Prohibits the Cycle Counter, PMCCNTR_EL0, from counting at EL3.

MCCD	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited at EL3.

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SBRBE, bits [33:32]
When FEAT_BRBE is implemented:

Secure Branch Record Buffer Enable. Controls branch recording by the BRBE, and access to BRBE registers and instructions at EL2 and EL1.

SBRBE	Meaning
0b00	Direct accesses to BRBE registers and instructions, except when in EL3, generate a Trap exception to EL3. EL0, EL1, and EL2 are prohibited regions.
0b01	Direct accesses to BRBE registers and instructions in Secure state, except when in EL3, generate a Trap exception to EL3. EL0, EL1, and EL2 in Secure state are prohibited regions. This control does not cause any direct accesses to BRBE registers when not in Secure state to be trapped, and does not cause any Exception levels when not in Secure state to be a prohibited region.
0b10	Direct accesses to BRBE registers and instructions, except when in EL3, generate a Trap exception to EL3. This control does not cause any Exception levels to be prohibited regions.
0b11	This control does not cause any direct accesses to BRBE registers or instruction to be trapped, and does not cause any Exception levels to be a prohibited region.

The Branch Record Buffer registers trapped by this control are: BRBCR_EL1, BRBCR_EL2, BRBCR_EL12, BRBFCR_EL1, BRBIDR0_EL1, BRBINF<n>_EL1, BRBINFINJ_EL1, BRBSRC<n>_EL1, BRBSRCINJ_EL1, BRBTGT<n>_EL1, BRBTGTINJ_EL1, and BRBTS_EL1.

The Branch Record Buffer instructions trapped by this control are:

- BRB IALL.
- BRB INJ.

Note

If FEAT_BRBEv1p1 is not implemented, EL3 is a prohibited region.

If EL3 is not implemented then the Effective value of this field is `0b11`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSSE, bits [31:30]
When FEAT_PMUv3_SS is implemented:

Performance Monitors Snapshot Enable. Controls the generation of Capture events.

PMSSE	Meaning
`0b00`	Capture events are disabled.
`0b01`	Capture events are controlled by MDCR_EL2.PMSSE.
`0b10`	Capture events are enabled and prohibited.
`0b11`	Capture events are enabled and allowed.

If EL3 is not implemented, then the Effective value of this field is `0b01`.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'00'`.

Otherwise:

Reserved, RES0.

Bit [29]

Reserved, RES0.

MTPME, bit [28]
When FEAT_MTPMU is implemented:

Multi-threaded PMU Enable. Enables use of the PMEVTYPER<n>_EL0.MT bits.

MTPME	Meaning
`0b0`	FEAT_MTPMU is disabled. The Effective value of PMEVTYPER<n>_EL0.MT is 0.
`0b1`	PMEVTYPER<n>_EL0.MT bits not affected by this field.

If FEAT_MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this field is 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'1'`.

Otherwise:

Reserved, RES0.

TDCC, bit [27]

When FEAT_FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel at EL2, EL1, and EL0 to EL3.

TDCC	Meaning
0b0	This control does not cause any register accesses to be trapped.
0b1	Accesses to the DCC registers at EL2, EL1, and EL0 generate a Trap exception to EL3, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state.

The DCC registers trapped by this control are:

AArch64: OSDTRRX_EL1, OSDTRTX_EL1, MDCCSR_EL0, MDCCINT_EL1, and, when the PE is in Non-debug state, DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0.

AArch32: DBGDTRRXext, DBGDTRTXext, DBGDSCRint, DBGDCCINT, and, when the PE is in Non-debug state, DBGDTRRXint and DBGDTRTXint.

The traps are reported with EC syndrome value:

- 0x05 for trapped AArch32 MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped AArch32 LDC to DBGDTRTXint and STC from DBGDTRRXint.
- 0x18 for trapped AArch64 MRS and MSR accesses.

When the PE is in Debug state, MDCR_EL3.TDCC does not trap any accesses to:

AArch64: DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0.

AArch32: DBGDTRRXint and DBGDTRTXint.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSTBE, bit [26]

When FEAT_TRBE is implemented and FEAT_RME is implemented:

Non-secure Trace Buffer Extended. Together with MDCR_EL3.NSTB, controls the trace buffer owning Security state and accesses to trace buffer System registers from EL2 and EL1.

For a description of the values derived by evaluating NSTB and NSTBE together, see MDCR_EL3.NSTB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSTB, bits [25:24]

When FEAT_TRBE is implemented and FEAT_RME is implemented:

Non-secure Trace Buffer. Together with MDCR_EL3.NSTBE, controls the trace buffer owning Security state and accesses to trace buffer System registers from EL2 and EL1.

NSTBE	NSTB	Meaning
0b0	0b00	Trace buffer owning Security state is Secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Realm and Non-secure states. Accesses to trace buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception. When Secure state is not implemented, this encoding is reserved.
0b0	0b01	Trace buffer owning Security state is Secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Realm and Non-secure states. Accesses to trace buffer System registers at Realm and Non-secure EL2, and Realm and Non-secure EL1, are trapped to EL3, unless the access generates a higher priority exception. When Secure state is not implemented, this encoding is reserved.
0b0	0b10	Trace buffer owning Security state is Non-secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Secure and Realm states. Accesses to trace buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
0b0	0b11	Trace buffer owning Security state is Non-secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Secure and Realm states. Accesses to trace buffer System registers at Secure and Realm EL2, and Secure and Realm EL1, are trapped to EL3, unless the access generates a higher priority exception.
0b1	0b10	Trace buffer owning Security state is Realm state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Secure and Non-secure states. Accesses to trace buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a highe priority exception.
0b1	0b11	Trace buffer owning Security state is Realm state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Secure and Non-secure states. Accesses to trace buffer System registers at Secure and Non-secure EL2, and Secure and Non-secure EL1, are trapped to EL3, unless the access generates a higher priority exception.

All other combinations of MDCR_EL3.NSTBE and MDCR_EL3.NSTB are reserved.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to TRBBASER_EL1, TRBLIMITR_EL1, TRBMAR_EL1, TRBPTR_EL1, TRBSR_EL1, and TRBTRG_EL1.
- If FEAT_TRBE_MPAM is implemented, MRS and MSR accesses to TRBMPAM_EL1.
- If FEAT_TRBE_EXC is implemented, MRS and MSR accesses to TRBSR_EL2 and TRBSR_EL12.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3, reported using EC syndrome value 0x18.

When TraceBufferEnabled()==FALSE, these controls have no effect on whether tracing is prohibited.

If the Trace Buffer Unit is enabled and using Self-hosted mode, and MDCR_EL3.{NSTB, NSTBE} selects a reserved value, then the behavior is CONSTRAINED UNPREDICTABLE, and the Trace Buffer Unit does one of:

- Behaves as if the Trace Buffer Unit is disabled.
- Selects an implemented Security state as the owning Security state.
- When trace data is received from the trace unit, it is not written to memory and the Trace Buffer Unit generates a trace buffer management event, as follows:
    - TRBSR_ELx.IRQ is set to 1.
    - If TRBSR_ELx.S is 0, then all of the following occur:
        - TRBSR_ELx.S is set to 1, Collection is stopped.
        - TRBSR_ELx.EC is set to 0x00, Other buffer management event.
        - TRBSR_ELx.BSC is set to 0b000000, Access not allowed.
    - The other fields in TRBSR_ELx are unchanged.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 1, then the Effective value of this field is 0b11.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 0b01.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TRBE is implemented:

Non-secure Trace Buffer. Controls the trace buffer owning Security state and accesses to trace buffer System registers from EL2 and EL1.

NSTB	Meaning
0b00	Trace buffer owning Security state is Secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Non-secure state. Accesses to trace buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
0b01	Trace buffer owning Security state is Secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Non-secure state. Accesses to trace buffer System registers at EL2 and EL1 in Non-secure state are trapped to EL3, unless the access generates a higher priority exception.
0b10	Trace buffer owning Security state is Non-secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Secure state. Accesses to trace buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
0b11	Trace buffer owning Security state is Non-secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Secure state. Accesses to trace buffer System registers at EL2 and EL1 in Secure state are trapped to EL3, unless the access generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to TRBBASER_EL1, TRBLIMITR_EL1, TRBMAR_EL1, TRBPTR_EL1, TRBSR_EL1, and TRBTRG_EL1.
- If FEAT_TRBE_MPAM is implemented, MRS and MSR accesses to TRBMPAM_EL1.
- If FEAT_TRBE_EXC is implemented, MRS and MSR accesses to TRBSR_EL2 and TRBSR_EL12.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3, reported using EC syndrome value 0x18.

When TraceBufferEnabled()==FALSE, this control has no effect on whether tracing is prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 1, then the Effective value of this field is 0b11.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 0b01.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCCD, bit [23]
When FEAT_PMUv3p5 is implemented:

Secure Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting in Secure state and EL3.

SCCD	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited in Secure state and EL3.

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

ETAD, bit [22]

When FEAT_RME is implemented, FEAT_TRC_EXT is implemented, and FEAT_TRBE is implemented:

External Trace Access Disable. Together with MDCR_EL3.ETADE, controls access to trace unit registers by an external debugger.

ETADE	ETAD	Meaning
0b0	0b0	No accesses from an external debugger to trace unit registers are prohibited by this control.
0b0	0b1	Realm and Non-secure accesses from an external debugger to trace unit registers are prohibited. Other accesses from an external debugger to trace unit registers are not affected by this control.
0b1	0b0	Secure and Non-secure accesses from an external debugger to trace unit registers are prohibited. Other accesses from an external debugger to trace unit registers are not affected by this control.
0b1	0b1	Secure, Non-secure, and Realm accesses from an external debugger to trace unit registers are prohibited. Other accesses from an external debugger to trace unit registers are not affected by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When FEAT_TRC_EXT is implemented and FEAT_TRBE is implemented:

External Trace Access Disable. Controls Non-secure access to trace unit registers by an external debugger.

ETAD	Meaning
0b0	No accesses from an external debugger to the trace unit registers are prohibited by this control.
0b1	Non-secure accesses from an external debugger to some trace unit registers are prohibited. See individual registers for the effect of this field.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

EPMAD, bit [21]

When FEAT_RME is implemented and FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable. Together with MDCR_EL3.EPMADE, controls access to Performance Monitor registers by an external debugger.

External accesses of the following Performance Monitor registers are affected by this control:

- PMCCFILTR_EL0, PMCCNTR_EL0, PMCFGR, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMINTENCLR_EL1, PMINTENSET_EL1, PMOVSCLR_EL0, and PMOVSSET_EL0.
- If PMEVFILT2R<n> is implemented, PMEVFILT2R<n>.
- If implemented, PMIIDR.
- If PMSWINC_EL0 is implemented in the external view, PMSWINC_EL0.
- If FEAT_PMUv3p4 is implemented, PMMIR.
- If FEAT_PMUv3_EXT64 is implemented, PMCNTEN, PMINTEN, and PMOVS.

- If FEAT_PMUv3p9 is implemented, PMZR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, PMCGCR0, PMICFILTR_EL0, and PMICNTR_EL0.
- If FEAT_PCSRv8p9 is implemented, PMPCSCTL.

EPMADE	EPMAD	Meaning
0b0	0b0	No accesses from an external debugger to affected Performance Monitor registers are prohibited by this control.
0b0	0b1	Realm and Non-secure accesses from an external debugger to affected Performance Monitor registers are prohibited. Other accesses from an external debugger to affected Performance Monitor registers are not affected by this control.
0b1	0b0	Secure and Non-secure accesses from an external debugger to affected Performance Monitor registers are prohibited. Other accesses from an external debugger to affected Performance Monitor registers are not affected by this control.
0b1	0b1	Secure, Non-secure, and Realm accesses from an external debugger to affected Performance Monitor registers are prohibited. Other accesses from an external debugger to affected Performance Monitor registers are not affected by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When FEAT_Debugv8p4 is implemented and FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable. Controls Non-secure access to Performance Monitor registers by an external debugger.

External accesses of the following Performance Monitor registers are affected by this control:

- PMCCFILTR_EL0, PMCCNTR_EL0, PMCFGR, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMINTENCLR_EL1, PMINTENSET_EL1, PMOVSCLR_EL0, and PMOVSSET_EL0.
- If PMEVFILT2R<n> is implemented, PMEVFILT2R<n>.
- If implemented, PMIIDR.
- If PMSWINC_EL0 is implemented in the external view, PMSWINC_EL0.
- If FEAT_PMUv3p4 is implemented, PMMIR.
- If FEAT_PMUv3_EXT64 is implemented, PMCNTEN, PMINTEN, and PMOVS.
- If FEAT_PMUv3p9 is implemented, PMZR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, PMCGCR0, PMICFILTR_EL0, and PMICNTR_EL0.
- If FEAT_PCSRv8p9 is implemented, PMPCSCTL.

EPMAD	Meaning
0b0	No accesses from an external debugger to the Performance Monitor registers are prohibited by this control.
0b1	Non-secure accesses from an external debugger to the affected Performance Monitor registers are prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable. Controls access to Performance Monitor registers by an external debugger.

External accesses of the following Performance Monitor registers are affected by this control:

- PMCCFILTR_EL0, PMCCNTR_EL0, PMCFGR, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMINTENCLR_EL1, PMINTENSET_EL1, PMOVSCLR_EL0, and PMOVSSET_EL0.
- If PMEVFILT2R<n> is implemented, PMEVFILT2R<n>.
- If implemented, PMIIDR.
- If PMSWINC_EL0 is implemented in the external view, PMSWINC_EL0.
- If FEAT_PMUv3p4 is implemented, PMMIR.

EPMAD	Meaning
0b0	No accesses from an external debugger to the Performance Monitor registers are prohibited by this control.
0b1	If the IMPLEMENTATION DEFINED authentication interface function `ExternalSecureInvasiveDebugEnabled()` returns FALSE, then accesses from an external debugger to the affected Performance Monitor registers are prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

EDAD, bit [20]
When FEAT_RME is implemented:

External Debug Access Disable. Together with MDCR_EL3.EDADE, controls access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger.

External accesses of the following debug registers are affected by this control:

- DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGWCR<n>_EL1, DBGWVR<n>_EL1, and OSLAR_EL1.

EDADE	EDAD	Meaning
0b0	0b0	No accesses from an external debugger to affected debug registers are prohibited by this control.
0b0	0b1	Realm and Non-secure accesses from an external debugger to affected debug registers are prohibited. Other accesses from an external debugger to affected debug registers are not affected by this control.
0b1	0b0	Secure and Non-secure accesses from an external debugger to affected debug registers are prohibited. Other accesses from an external debugger to affected debug registers are not affected by this control.
0b1	0b1	Secure, Non-secure, and Realm accesses from an external debugger to affected debug registers are prohibited. Other accesses from an external debugger to affected debug registers are not affected by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

When FEAT_Debugv8p4 is implemented:

External Debug Access Disable. Controls Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger.

External accesses of the following debug registers are affected by this control:

- DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGWCR<n>_EL1, DBGWVR<n>_EL1, and OSLAR_EL1.

EDAD	Meaning
0b0	No accesses from an external debugger to the debug registers are prohibited by this control.
0b1	Non-secure accesses from an external debugger to the affected debug registers are prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When FEAT_Debugv8p2 is implemented:

External Debug Access Disable. Controls access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger.

External accesses of the following debug registers are affected by this control:

- DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGWCR<n>_EL1, DBGWVR<n>_EL1, and OSLAR_EL1.

EDAD	Meaning
0b0	No accesses from an external debugger to the debug registers are prohibited by this control.
0b1	If the IMPLEMENTATION DEFINED authentication interface function ExternalSecureInvasiveDebugEnabled() returns FALSE, then accesses from an external debugger to the affected debug registers are prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

External Debug Access Disable. Controls access to breakpoint registers, watchpoint registers, and optionally OSLAR_EL1 by an external debugger.

External accesses of the following debug registers are affected by this control:

- DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGWCR<n>_EL1, and DBGWVR<n>_EL1.
- It is IMPLEMENTATION DEFINED whether this control affects OSLAR_EL1.

EDAD	Meaning
0b0	No accesses from an external debugger to the debug registers are prohibited by this control.
0b1	If the IMPLEMENTATION DEFINED authentication interface function ExternalSecureInvasiveDebugEnabled() returns FALSE, then accesses from an external debugger to the affected debug registers are prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TTRF, bit [19]

When FEAT_TRF is implemented:

Trap Trace Filter controls. Traps use of the Trace Filter control registers at EL2 and EL1 to EL3.

The Trace Filter registers trapped by this control are:

- TRFCR_EL2, TRFCR_EL12, TRFCR_EL1, reported using EC syndrome value 0x18.
- HTRFCR and TRFCR, reported using EC syndrome value 0x03.

TTRF	Meaning
0b0	Accesses to Trace Filter registers at EL2 and EL1 are not affected by this field.
0b1	Accesses to Trace Filter registers at EL2 and EL1 generate a Trap exception to EL3, unless the access generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

STE, bit [18]
When FEAT_TRF is implemented and Secure state is implemented:

Secure Trace enable. Enables tracing in Secure state.

STE	Meaning
0b0	Trace prohibited in Secure state unless overridden by the IMPLEMENTATION DEFINED authentication interface.
0b1	Trace in Secure state is not affected by this field.

This field also controls the level of authentication required by an external debugger to enable external tracing. See 'Register controls to enable self-hosted trace'.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SPME, bit [17]
When FEAT_PMUv3 is implemented and FEAT_PMUv3p7 is implemented:

Secure Performance Monitors Enable. Controls PMU operation in Secure state and at EL3 when MDCR_EL3.MPMX is 0.

SPME	Meaning
0b0	Affected counters are prohibited from counting in Secure state and at EL3. If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled in Secure state and at EL3. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.
0b1	Counters are not affected by this mechanism.

When MDCR_EL3.MPMX is 0, the counters affected by this field are:

- Event counters in the first and second ranges. For more information about event counter ranges, see MDCR_EL2.HPMN.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter, PMICNTR_EL0.
- If PMCR_EL0.DP is 1, the cycle counter, PMCCNTR_EL0.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

When MDCR_EL3.MPMX is 1, this field controls which event counters are affected by MDCR_EL3.MPMX at EL3. See MDCR_EL3.MPMX for more information.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When FEAT_PMUv3 is implemented and FEAT_Debugv8p2 is implemented:

Secure Performance Monitors Enable. Controls PMU operation in Secure state.

SPME	Meaning
0b0	Event counting is prohibited in Secure state. If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled in Secure state. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.
0b1	Event counting and PMCCNTR_EL0 are not affected by this mechanism.

The counters affected by this field are:

- All event counters.
- If PMCR_EL0.DP is 1, the cycle counter, PMCCNTR_EL0.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When FEAT_PMUv3 is implemented:

Secure Performance Monitors Enable. Controls PMU operation in Secure state.

SPME	Meaning
0b0	If ExternalSecureNoninvasiveDebugEnabled() is FALSE, then all the following apply: • Event counting is prohibited in Secure state. • If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled in Secure state. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.
0b1	Event counting and PMCCNTR_EL0 are not affected by this mechanism.

If ExternalSecureNoninvasiveDebugEnabled() is TRUE then the event counters and PMCCNTR_EL0 are not affected by this field.

Otherwise, the counters affected by this field are:

- All event counters.
- If PMCR_EL0.DP is 1, the cycle counter, PMCCNTR_EL0.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SDD, bit [16]

When Secure state is implemented:

AArch64 Secure Self-hosted invasive debug disable. Disables Software debug exceptions in Secure state, other than Breakpoint Instruction exceptions.

SDD	Meaning
0b0	Debug exceptions in Secure state are not affected by this field.
0b1	Debug exceptions, other than Breakpoint Instruction exceptions, are disabled from all Exception levels in Secure state.

The SDD bit is ignored unless both of the following are true:

- The PE is in Secure state.
- The Effective value of SCR_EL3.RW is 0.

If Secure EL2 is implemented and enabled, and Secure EL1 is using AArch32, then:

- If debug exceptions from Secure EL1 are enabled, debug exceptions from Secure EL0 are also enabled.
- Otherwise, debug exceptions from Secure EL0 are enabled only if the value of SDER32_EL3.SUIDEN is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPD32, bits [15:14]
When FEAT_AA32EL1 is implemented:

AArch32 Secure self-hosted privileged debug. Enables or disables debug exceptions from Secure EL1 using AArch32, other than Breakpoint Instruction exceptions.

SPD32	Meaning
0b00	Legacy mode. Debug exceptions from Secure EL1 are enabled by the IMPLEMENTATION DEFINED authentication interface.
0b10	Secure privileged debug disabled. Debug exceptions from Secure EL1 are disabled.
0b11	Secure privileged debug enabled. Debug exceptions from Secure EL1 are enabled.

Other values are reserved, and have the CONSTRAINED UNPREDICTABLE behavior that they must have the same behavior as 0b00. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

This field has no effect on Breakpoint Instruction exceptions. These are always enabled.

This field is ignored unless both of the following are true:

- The PE is in Secure state.
- The Effective value of SCR_EL3.RW is 0.

If Secure EL1 is using AArch32, then:

- If debug exceptions from Secure EL1 are enabled, then debug exceptions from Secure EL0 are also enabled.
- Otherwise, debug exceptions from Secure EL0 are enabled only if the value of SDER32_EL3.SUIDEN is 1.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSPB, bits [13:12]

When FEAT_SPE is implemented and FEAT_RME is implemented:

Non-secure Profiling Buffer. Together with MDCR_EL3.NSPBE, controls the Profiling Buffer owning Security state and accesses to Statistical Profiling and Profiling Buffer System registers from EL2 and EL1.

NSPBE	NSPB	Meaning
0b0	0b00	Profiling Buffer owning Security state is Secure state. Profiling is disabled in Realm and Non-secure states. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception. When Secure state is not implemented, this encoding is reserved.
0b0	0b01	Profiling Buffer owning Security state is Secure state. Profiling is disabled in Realm and Non-secure states. Accesses to Statistical Profiling and Profiling Buffer System registers at Realm and Non-secure EL2, and Realm and Non-secure EL1, are trapped to EL3, unless the access generates a higher priority exception. When Secure state is not implemented, this encoding is reserved.
0b0	0b10	Profiling Buffer owning Security state is Non-secure state. Profiling is disabled in Secure and Realm states. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
0b0	0b11	Profiling Buffer owning Security state is Non-secure state. Profiling is disabled in Secure and Realm states. Accesses to Statistical Profiling and Profiling Buffer System registers at Secure and Realm EL2, and Secure and Realm EL1, are trapped to EL3, unless the access generates a higher priority exception.
0b1	0b10	Profiling Buffer owning Security state is Realm state. Profiling is disabled in Secure and Non-secure states. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a highe priority exception.
0b1	0b11	Profiling Buffer owning Security state is Realm state. Profiling is disabled in Secure and Non-secure states. Accesses to Statistical Profiling and Profiling Buffer System registers at Secure and Non-secure EL2, and Secure and Non-secure EL1, are trapped to EL3, unless the access generates a higher priority exception.

All other combinations of MDCR_EL3.NSPBE and MDCR_EL3.NSPB are reserved.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to PMBLIMITR_EL1, PMBPTR_EL1, PMBSR_EL1, PMSCR_EL1, PMSCR_EL2, PMSCR_EL12, PMSEVFR_EL1, PMSFCR_EL1, PMSICR_EL1, PMSIRR_EL1, and PMSLATFR_EL1.
- MRS accesses to PMSIDR_EL1.
- If FEAT_SPE_FnE is implemented, MRS and MSR accesses to PMSNEVFR_EL1.
- If FEAT_SPE_FDS is implemented, MRS and MSR accesses to PMSDSFR_EL1.
- If FEAT_SPE_EXC is implemented, MRS and MSR accesses to PMBSR_EL2 and PMBSR_EL12.
- If FEAT_SPE_nVM is implemented, MRS and MSR accesses to PMBMAR_EL1.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3, reported using EC syndrome value 0x18.

If profiling is enabled and MDCR_EL3.{NSPB, NSPBE} selects a reserved value, then the behavior is CONSTRAINED UNPREDICTABLE, and the Statistical Profiling Unit does one of:

- Behaves as if profiling is disabled.
- Selects an implemented Security state as the owning Security state.
- When profiling data is generated, it is not written to memory and the Statistical Profiling Unit generates a Profiling Buffer management event, as follows:
  - If PMBSR_ELx.S is 0, then all of the following occur:
    - PMBSR_ELx.S is set to 1.
    - PMBSR_ELx.DL is set to 1.
    - PMBSR_ELx.EC is set to 0b000000, Other buffer management event.
    - PMBSR_ELx.BSC is set to 0b000000, Access not allowed.
  - The other fields in PMBSR_ELx are unchanged.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 1, then the Effective value of this field is 0b11.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 0b01.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPE is implemented:

Non-secure Profiling Buffer. Controls the Profiling Buffer owning Security state and accesses to Statistical Profiling and Profiling Buffer System registers from EL2 and EL1.

NSPB	Meaning
0b00	Profiling Buffer owning Security state is Secure state. Profiling is disabled in Non-secure state. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
0b01	Profiling Buffer owning Security state is Secure state. Profiling is disabled in Non-secure state. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 in Non-secure state are trapped to EL3, unless the access generates a higher priority exception.
0b10	Profiling Buffer owning Security state is Non-secure state. Profiling is disabled in Secure state. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
0b11	Profiling Buffer owning Security state is Non-secure state. Profiling is disabled in Secure state. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 in Secure state are trapped to EL3, unless the access generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to PMBLIMITR_EL1, PMBPTR_EL1, PMBSR_EL1, PMSCR_EL1, PMSCR_EL2, PMSCR_EL12, PMSEVFR_EL1, PMSFCR_EL1, PMSICR_EL1, PMSIRR_EL1, and PMSLATFR_EL1.
- MRS accesses to PMSIDR_EL1.
- If FEAT_SPE_FnE is implemented, MRS and MSR accesses to PMSNEVFR_EL1.
- If FEAT_SPE_FDS is implemented, MRS and MSR accesses to PMSDSFR_EL1.
- If FEAT_SPE_EXC is implemented, MRS and MSR accesses to PMBSR_EL2 and PMBSR_EL12.
- If FEAT_SPE_nVM is implemented, MRS and MSR accesses to PMBMAR_EL1.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3, reported using EC syndrome value 0x18.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 1, then the Effective value of this field is 0b11.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 0b01.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSPBE, bit [11]
When FEAT_SPE is implemented and FEAT_RME is implemented:

Non-secure Profiling Buffer Extended. Together with MDCR_EL3.NSPB, controls the Profiling Buffer owning Security state and accesses to Statistical Profiling and Profiling Buffer System registers from EL2 and EL1.

For a description of the values derived by evaluating NSPB and NSPBE together, see MDCR_EL3.NSPB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TDOSA, bit [10]

When FEAT_DoubleLock is implemented:

Trap debug OS-related register access. Traps EL2 and EL1 System register accesses to the powerdown debug registers to EL3.

Accesses to the registers are trapped as follows:

- Accesses from AArch64 state, OSLAR_EL1, OSLSR_EL1, OSDLR_EL1, DBGPRCR_EL1, and any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this field, are trapped to EL3 and reported using EC syndrome value `0x18`.
- Accesses using MCR or MRC to DBGOSLAR, DBGOSLSR, DBGOSDLR, and DBGPRCR, are trapped to EL3 and reported using EC syndrome value `0x05`.
- Accesses to any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this field.

TDOSA	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL2 and EL1 System register accesses to the powerdown debug registers are trapped to EL3, unless it is trapped by HDCR.TDOSA or MDCR_EL2.TDOSA.

Note

The powerdown debug registers are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trap debug OS-related register access. Traps EL2 and EL1 System register accesses to the powerdown debug registers to EL3.

The following registers are affected by this trap:

- AArch64: OSLAR_EL1, OSLSR_EL1, and DBGPRCR_EL1.
- AArch32: DBGOSLAR, DBGOSLSR, and DBGPRCR.
- AArch64 and AArch32: Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this field.
- It is IMPLEMENTATION DEFINED whether accesses to OSDLR_EL1 and DBGOSDLR are trapped.

TDOSA	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL2 and EL1 System register accesses to the powerdown debug registers are trapped to EL3, unless it is trapped by HDCR.TDOSA or MDCR_EL2.TDOSA.

Note

The powerdown debug registers are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDA, bit [9]

Trap accesses of debug System registers. Enables a trap to EL3 on accesses of debug System registers.

TDA	Meaning
0b0	Accesses of the specified debug System registers are not trapped by this mechanism.
0b1	Accesses of the specified debug System registers at EL2, EL1, and EL0 are trapped to EL3, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to DBGAUTHSTATUS_EL1, DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGCLAIMCLR_EL1, DBGCLAIMSET_EL1, DBGVCR32_EL2, DBGWCR<n>_EL1, DBGWVR<n>_EL1, MDCCINT_EL1, MDCCSR_EL0, MDCR_EL2, MDRAR_EL1, MDSCR_EL1, OSDTRRX_EL1, OSDTRTX_EL1, and OSECCR_EL1.
- If FEAT_Debugv8p9 is implemented, MRS and MSR accesses to MDSELR_EL1.
- If FEAT_STEP2 is implemented, MRS and MSR accesses to MDSTEPOP_EL1.
- In Non-debug state, MRS accesses to DBGDTRRX_EL0 and DBGDTR_EL0 and MSR accesses to DBGDTRTX_EL0 and DBGDTR_EL0.

In AArch32 state, the instructions affected by this control are:

- MRC and MCR accesses to DBGAUTHSTATUS, DBGBCR<n>, DBGBVR<n>, DBGBXVR<n>, DBGCLAIMCLR, DBGCLAIMSET, DBGDCCINT, DBGDEVID, DBGDEVID1, DBGDEVID2, DBGDIDR, DBGDRAR, DBGDSAR, DBGDSCRext, DBGDSCRint, DBGDTRRXext, DBGDTRTXext, DBGOSECCR, DBGVCR, DBGWCR<n>, DBGWFAR, DBGWVR<n>, HDCR, and SDER.
- MRRC accesses to DBGDRAR and DBGDSAR.
- STC accesses to DBGDTRRXint and LDC accesses to DBGDTRTXint.
- In Non-debug state, MRC accesses to DBGDTRRXint and MCR accesses to DBGDTRTXint.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Trapped AArch32 instructions are reported using EC syndrome value 0x03 for MRC and MCR accesses with coproc == 0b1111, 0x05 for MCR and MCR accesses with coproc == 0b1110, 0x06 for LDC and STC accesses, and 0x0C for MRRC accesses.

The following instructions are not trapped in Debug state:

- AArch64 MRS accesses to DBGDTRRX_EL0 and DBGDTR_EL0 and MSR accesses to DBGDTRTX_EL0 and DBGDTR_EL0.
- AArch32 MRC accesses to DBGDTRRXint and MCR accesses to DBGDTRTXint.

If 16 or fewer breakpoints and 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI, then it is IMPLEMENTATION DEFINED whether AArch64 accesses to MDSELR_EL1 are trapped to EL3 when MDCR_EL3.TDA is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

EnPM2, bit [7]

When FEAT_PMUv3p9 is implemented, or FEAT_SPMU is implemented, or FEAT_EBEP is implemented, or FEAT_PMUv3_SS is implemented, or FEAT_SPMU2 is implemented:

Enable access to PMU registers. When disabled, accesses to PMU registers generate a trap to EL3.

EnPM2	Meaning
0b0	Accesses of the specified PMU registers at EL2, EL1, and EL0 are trapped to EL3, unless the instruction generates a higher priority exception. If FEAT_PMUv3_ICNTR is implemented, then: • PMCNTENCLR_EL0.F0, PMCNTENSET_EL0.F0, PMOVSCLR_EL0.F0, PMOVSSET_EL0.F0, and PMZR_EL0.F0 read-as-zero and ignore writes at EL2, EL1, and EL0. • PMINTENCLR_EL1.F0 and PMINTENSET_EL1.F0 read-as-zero and ignore writes at EL2 and EL1.
0b1	Accesses of the specified PMU registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are:

- If FEAT_EBEP is implemented or FEAT_PMUv3_SS is implemented, MRS and MSR accesses to PMECR_EL1.
- If FEAT_PMUv3_ICNTR is implemented, MRS and MSR accesses to PMICFILTR_EL0 and PMICNTR_EL0.
- If FEAT_PMUv3p9 is implemented, MRS and MSR accesses to PMUACR_EL1.
- If FEAT_SEBEP is implemented, MRS and MSR accesses to PMIAR_EL1.
- If FEAT_SPMU is implemented, MRS and MSR accesses to SPMACCESSR_EL1, SPMACCESSR_EL2, SPMACCESSR_EL12, SPMCFGR_EL1, SPMCGCR<n>_EL1, SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMDEVAFF_EL1, SPMDEVARCH_EL1, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMIIDR_EL1, SPMINTENCLR_EL1, SPMINTENSET_EL1, SPMOVSCLR_EL0, SPMOVSSET_EL0, SPMSCR_EL1, and SPMSELR_EL0.
- If FEAT_SPMU2 is implemented, MSR writes to SPMZR_EL0.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPM, bit [6]

When FEAT_PMUv3 is implemented:

Trap accesses of PMU registers. Enables a trap to EL3 on accesses of PMU registers.

TPM	Meaning
0b0	Accesses of the specified PMU registers are not trapped by this mechanism.
0b1	Accesses of the specified PMU registers at EL2, EL1, and EL0 are trapped to EL3, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to PMCCFILTR_EL0, PMCCNTR_EL0, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMINTENCLR_EL1, PMINTENSET_EL1, PMOVSCLR_EL0, PMOVSSET_EL0, PMSELR_EL0, PMSWINC_EL0, PMUSERENR_EL0, PMXEVCNTR_EL0, and PMXEVTYPER_EL0.
- MRS accesses to PMCEID0_EL0 and PMCEID1_EL0.
- If FEAT_PMUv3p4 is implemented, MRS accesses to PMMIR_EL1.
- If FEAT_PMUv3p9 is implemented, MSR accesses to PMZR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, MRS and MSR accesses to PMICFILTR_EL0 and PMICNTR_EL0.
- If FEAT_EBEP is implemented or FEAT_PMUv3_SS is implemented, MRS and MSR accesses to PMECR_EL1.
- If FEAT_SEBEP is implemented, MRS and MSR accesses to PMIAR_EL1.

In AArch32 state, the instructions affected by this control are:

- MRC and MCR accesses to PMCCFILTR, PMCCNTR, PMCEID0, PMCEID1, PMCNTENCLR, PMCNTENSET, PMCR, PMEVCNTR<n>, PMEVTYPER<n>, PMINTENCLR, PMINTENSET, PMOVSR, PMOVSSET, PMSELR, PMSWINC, PMUSERENR, PMXEVCNTR, and PMXEVTYPER.
- MRRC and MCRR accesses to PMCCNTR.
- If FEAT_PMUv3p1 is implemented, MRC accesses to PMCEID2 and PMCEID3.
- If FEAT_PMUv3p4 is implemented, MRC accesses to PMMIR.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Trapped AArch32 instructions are reported using EC syndrome value 0x03 for MRC and MCR accesses, and 0x04 for MRRC and MCRR accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [5]

Reserved, RES0.

EDADE, bit [4]

When FEAT_RME is implemented:

External Debug Access Disable Extended. Together with MDCR_EL3.EDAD, controls access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger.

For a description of the values derived by evaluating MDCR_EL3.EDAD and MDCR_EL3.EDADE together, see MDCR_EL3.EDAD.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

ETADE, bit [3]

When FEAT_RME is implemented, FEAT_TRC_EXT is implemented, and FEAT_TRBE is implemented:

External Trace Access Disable Extended. Together with MDCR_EL3.ETAD, controls access to trace unit registers by an external debugger.

For a description of the values derived by evaluating MDCR_EL3.ETAD and MDCR_EL3.ETADE together, see MDCR_EL3.ETAD.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

EPMADE, bit [2]

When FEAT_RME is implemented and FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable Extended. Together with MDCR_EL3.EPMAD, controls access to Performance Monitor registers by an external debugger.

For a description of the values derived by evaluating MDCR_EL3.EPMAD and MDCR_EL3.EPMADE together, see MDCR_EL3.EPMAD.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

Bit [1]

Reserved, RES0.

RLTE, bit [0]

When FEAT_RME is implemented and FEAT_TRF is implemented:

Realm Trace enable. Enables tracing in Realm state.

RLTE	Meaning
0b0	Trace prohibited in Realm state, unless overridden by the IMPLEMENTATION DEFINED authentication interface.
0b1	Trace in Realm state is not affected by this field.

This field also controls the level of authentication that is required by an external debugger to enable external tracing.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Accessing MDCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0011	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDCR_EL3;
```

```
MSR MDCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0011	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.MDCR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDRAR_EL1, Monitor Debug ROM Address Register

The MDRAR_EL1 characteristics are:

Purpose

Defines the base physical address of a 4KB-aligned memory-mapped debug component, usually a ROM table that locates and describes the memory-mapped debug components in the system. Use of this register is deprecated.

Configuration

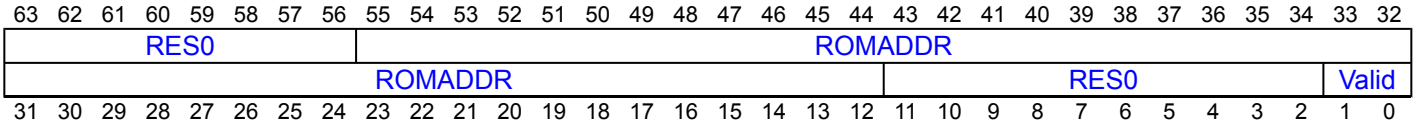
AArch64 System register MDRAR_EL1 bits [63:0] are architecturally mapped to AArch32 System register DBGDRAR[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MDRAR_EL1 are UNDEFINED.

Attributes

MDRAR_EL1 is a 64-bit register.

Field descriptions

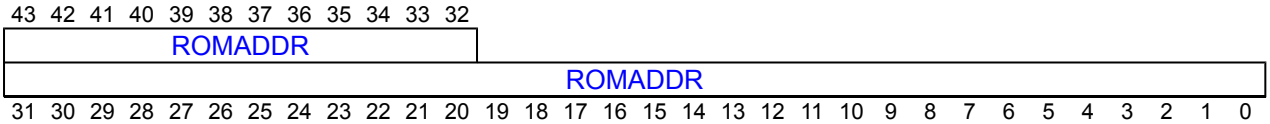


Bits [63:56]

Reserved, RES0.

ROMADDR, bits [55:12]

ROMADDR encoding when FEAT_D128 is implemented and MDRAR_EL1.Valid != 0b00



ROMADDR, bits [43:0]

Bits [55:12] of the ROM table physical address.

Bits [11:0] of the ROM table physical address are zero.

For implementations with fewer than 56 physical address bits, the corresponding upper bits of this field are RES0

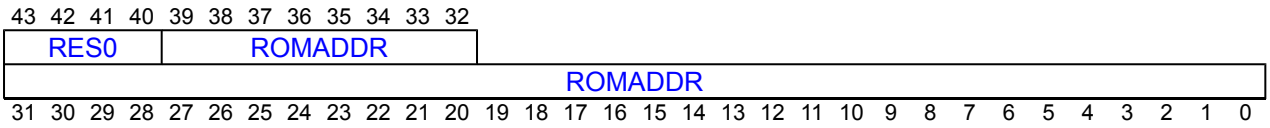
In an implementation that includes EL3, ROMADDR is an address in Non-secure PA space. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure PA space. If FEAT_RME is implemented, it is IMPLEMENTATION DEFINED whether the ROM table is also accessible in the Root or Realm PA spaces.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system where the implementation only supports execution in AArch32 state.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ROMADDR encoding when FEAT_D128 is not implemented, FEAT_LPA is implemented, and MDRAR_EL1.Valid != 0b00



Bits [43:40]

Reserved, RES0.

ROMADDR, bits [39:0]

Bits [51:12] of the ROM table physical address.

Bits [11:0] of the ROM table physical address are zero.

For implementations with fewer than 52 physical address bits, the corresponding upper bits of this field are RES0

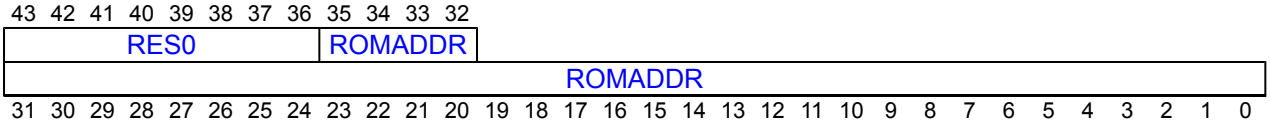
In an implementation that includes EL3, ROMADDR is an address in Non-secure PA space. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure PA space. If FEAT_RME is implemented, it is IMPLEMENTATION DEFINED whether the ROM table is also accessible in the Root or Realm PA spaces.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system where the implementation only supports execution in AArch32 state.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ROMADDR encoding when FEAT_D128 is not implemented, FEAT_LPA is not implemented, and MDRAR_EL1.Valid != 0b00



Bits [43:36]

Reserved, RES0.

ROMADDR, bits [35:0]

Bits [39:12] of the ROM table physical address.

Bits [11:0] of the ROM table physical address are zero.

For implementations with fewer than 48 physical address bits, the corresponding upper bits of this field are RES0

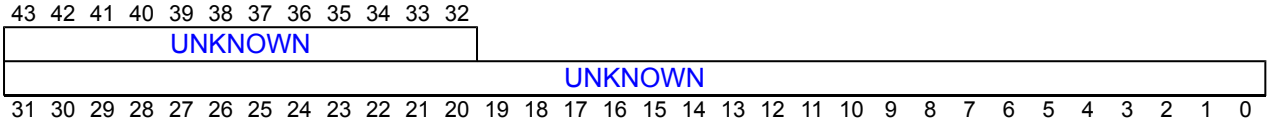
In an implementation that includes EL3, ROMADDR is an address in Non-secure PA space. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure PA space. If FEAT_RME is implemented, it is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Root or Realm PA spaces.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system where the implementation only supports execution in AArch32 state.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ROMADDR encoding when MDRAR_EL1.Valid == 0b00



Bits [43:0]

Reserved, UNKNOWN.

Bits [11:2]

Reserved, RES0.

Valid, bits [1:0]

This field indicates whether the ROM Table address is valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Valid	Meaning
0b00	ROM Table address is not valid. Software must ignore ROMADDR.
0b11	ROM Table address is valid.

Other values are reserved.

Arm recommends implementations set this field to zero.

Access to this field is **RO**.

Accessing MDRAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDRAR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDRA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDRAR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDRAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDRAR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDSCR_EL1, Monitor Debug System Control Register

The MDSCR_EL1 characteristics are:

Purpose

Main control register for the debug implementation.

Configuration

AArch64 System register MDSCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDSCRext[31:0].

AArch64 System register MDSCR_EL1 bit [15] is architecturally mapped to AArch32 System register DBGDSCRint[15].

AArch64 System register MDSCR_EL1 bit [12] is architecturally mapped to AArch32 System register DBGDSCRint[12].

AArch64 System register MDSCR_EL1 bits [5:2] are architecturally mapped to AArch32 System register DBGDSCRint[5:2].

AArch64 System register MDSCR_EL1 bits [31:29, 27:26, 23:21, 19, 14, 6] are architecturally mapped to External register EDSCR[31:29, 27:26, 23:21, 19, 14, 6].

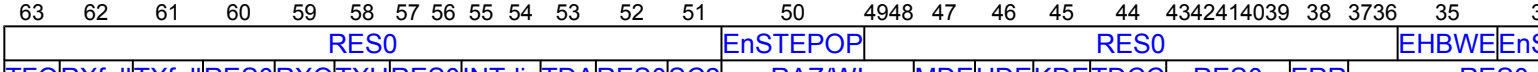
AArch64 System register MDSCR_EL1 bits [35, 33] are architecturally mapped to External register EDSCR2[3, 1].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MDSCR_EL1 are UNDEFINED.

Attributes

MDSCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:51]

Reserved, RES0.

EnSTEPOP, bit [50]

When FEAT_STEP2 is implemented:

Software step control bit. Enable execution from MDSTEPOP_EL1. Permitted values are:

EnSTEPOP	Meaning
0b0	Execution from MDSTEPOP_EL1 is disabled.
0b1	Execution from MDSTEPOP_EL1 is not disabled by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [49:36]

Reserved, RES0.

EHBWE, bit [35]

When FEAT_Debugv8p9 is implemented:

Extended Halting Breakpoint and Watchpoint Enable. Used for save/restore of EDSCR2.EHBWE.

When OSLSR_EL1.OSLK is 0, software must treat this field as UNK/SBZP.

When OSLSR_EL1.OSLK is 1, this field holds the value of EDSCR2.EHBWE. Reads and writes of this field are indirect accesses to EDSCR2.EHBWE.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

EnSPM, bit [34]

When FEAT_SPMU is implemented:

Enable access to System PMU registers. When disabled, accesses to System PMU registers generate a trap to EL1.

EnSPM	Meaning
`0b0`	Accesses of the specified System PMU registers at EL0 are trapped to EL1, unless the instruction generates a higher priority exception.
`0b1`	Accesses of the specified System PMU registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: `MRS` and `MSR` accesses to SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMOVSCLR_EL0, SPMOVSSET_EL0, and SPMSELR_EL0.

Unless the instruction generates a higher priority exception:

- If EL2 is implemented and enabled in the current Security state, and HCR_EL2.TGE is 1, then trapped instructions generate an exception to EL2.
- Otherwise, trapped instructions generate an exception to EL1.

Trapped instructions are reported using EC syndrome value `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTA, bit [33]

When FEAT_TRBE_EXT is implemented or FEAT_ETEv1p3 is implemented:

Trap Trace Accesses. Used for save/restore of EDSCR2.TTA.

When OSLSR_EL1.OSLK is 0, software must treat this field as UNK/SBZP.

When OSLSR_EL1.OSLK is 1, this field holds the value of EDSCR2.TTA. Reads and writes of this field are indirect accesses to EDSCR2.TTA.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

EMBWE, bit [32]

When FEAT_Debugv8p9 is implemented:

Extended Monitor Breakpoint and Watchpoint Enable. Enables use of additional breakpoints or watchpoints.

EMBWE	Meaning
`0b0`	Breakpoint and Watchpoint exceptions are disabled for each breakpoint <n> and watchpoint <n>, where n is greater than or equal to 16. The Effective value of MDSELR_EL1.BANK is zero at EL1.
`0b1`	Breakpoint and Watchpoint exceptions are not affected by this mechanism. The Effective value of MDSELR_EL1.BANK is not affected by this field.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and MDCR_EL3.EBWE is 0.
- EL2 is implemented and enabled in the current Security state, and MDCR_EL2.EBWE is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TFO, bit [31]

When FEAT_TRF is implemented:

Trace Filter override. Used for save/restore of EDSCR.TFO.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.TFO. Reads and writes of this bit are indirect accesses to EDSCR.TFO.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

Otherwise:

Reserved, RES0.

RXfull, bit [30]

Used for save/restore of EDSCR.RXfull.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.RXfull. Reads and writes of this bit are indirect accesses to EDSCR.RXfull.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

TXfull, bit [29]

Used for save/restore of EDSCR.TXfull.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.TXfull. Reads and writes of this bit are indirect accesses to EDSCR.TXfull.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

Bit [28]

Reserved, RES0.

RXO, bit [27]

Used for save/restore of EDSCR.RXO.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.RXO. Reads and writes of this bit are indirect accesses to EDSCR.RXO.

When OSLSR_EL1.OSLK == 1, if bits [27,6] of the value written to MDSCR_EL1 are {1,0}, that is, the RXO bit is 1 and the ERR bit is 0, the PE sets EDSCR.{RXO,ERR} to UNKNOWN values.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

TXU, bit [26]

Used for save/restore of EDSCR.TXU.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.TXU. Reads and writes of this bit are indirect accesses to EDSCR.TXU.

When OSLSR_EL1.OSLK == 1, if bits [26,6] of the value written to MDSCR_EL1 are {1,0}, that is, the TXU bit is 1 and the ERR bit is 0, the PE sets EDSCR.{TXU,ERR} to UNKNOWN values.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

Bits [25:24]

Reserved, RES0.

INTdis, bits [23:22]

Used for save/restore of EDSCR.INTdis.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this field holds the value of EDSCR.INTdis. Reads and writes of this field are indirect accesses to EDSCR.INTdis.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'00'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

TDA, bit [21]

Used for save/restore of EDSCR.TDA.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.TDA. Reads and writes of this bit are indirect accesses to EDSCR.TDA.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

Bit [20]

Reserved, RES0.

SC2, bit [19]

When FEAT_PCSRv8 is implemented, FEAT_VHE is implemented, and FEAT_PCSRv8p2 is not implemented:

Used for save/restore of EDSCR.SC2.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.SC2. Reads and writes of this bit are indirect accesses to EDSCR.SC2.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [18:16]

Reserved, RAZ/WI.

Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register.

MDE, bit [15]

Monitor debug events. Enable Breakpoint, Watchpoint, and Vector Catch exceptions.

MDE	Meaning
`0b0`	Breakpoint, Watchpoint, and Vector Catch exceptions disabled.
`0b1`	Breakpoint, Watchpoint, and Vector Catch exceptions enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HDE, bit [14]

Used for save/restore of EDSCR.HDE.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.HDE. Reads and writes of this bit are indirect accesses to EDSCR.HDE.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

KDE, bit [13]

Local (kernel) debug enable. If EL_D is using AArch64, enable debug exceptions within EL_D. Permitted values are:

KDE	Meaning
0b0	Debug exceptions, other than Breakpoint Instruction exceptions, disabled within EL_D.
0b1	All debug exceptions enabled within EL_D.

RES0 if EL_D is using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDCC, bit [12]

Traps EL0 accesses to the Debug Communication Channel (DCC) registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states, as follows:

- In AArch64 state, MRS or MSR accesses to the following DCC registers are trapped, reported using EC syndrome value 0x18:
  - MDCCSR_EL0.
  - If not in Debug state, DBGDTR_EL0, DBGDTRTX_EL0, and DBGDTRRX_EL0.
- In AArch32 state, MRC or MCR accesses to the following registers are trapped, reported using EC syndrome value 0x05.
  - DBGDSCRint, DBGDIDR, DBGDSAR, DBGDRAR.
  - If not in Debug state, DBGDTRRXint, and DBGDTRTXint.
- In AArch32 state, LDC access to DBGDTRRXint and STC access to DBGDTRTXint are trapped, reported using EC syndrome value 0x06.
- In AArch32 state, MRRC accesses to DBGDSAR and DBGDRAR are trapped, reported using EC syndrome value 0x0C.

TDCC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0 using AArch64: EL0 accesses to the AArch64 DCC registers are trapped. EL0 using AArch32: EL0 accesses to the AArch32 DCC registers are trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:7]

Reserved, RES0.

ERR, bit [6]

Used for save/restore of EDSCR.ERR.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.ERR. Reads and writes of this bit are indirect accesses to EDSCR.ERR.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

Bits [5:1]

Reserved, RES0.

SS, bit [0]

Software step control bit. If EL_D is using AArch64, enable Software step. Permitted values are:

SS	Meaning
0b0	Software step disabled
0b1	Software step enabled.

RES0 if EL_D is using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MDSCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDSCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.MDSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x158];
    else
        X[t, 64] = MDSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDSCR_EL1;
```

```
MSR MDSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.MDSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x158] = X[t, 64];
    else
        MDSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MDSCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDSELR_EL1, Breakpoint and Watchpoint Selection Register

The MDSELR_EL1 characteristics are:

Purpose

Selects the current breakpoints or watchpoints accessed by System register instructions.

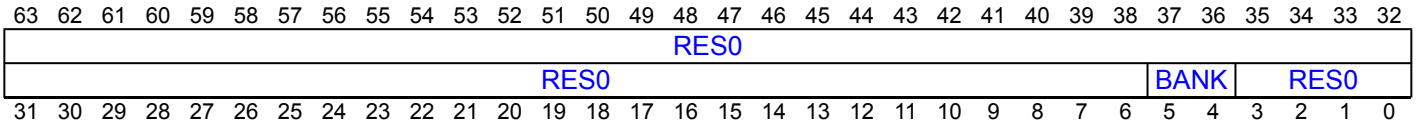
Configuration

This register is present only when FEAT_Debugv8p9 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MDSELR_EL1 are UNDEFINED.

Attributes

MDSELR_EL1 is a 64-bit register.

Field descriptions



Bits [63:6]

Reserved, RES0.

BANK, bits [5:4]

Breakpoint and watchpoint bank select.

BANK	Meaning	Applies when
0b00	Select 0 to 15.	
0b01	Select 16 to 31.	When NUM_BREAKPOINTS > 16 or NUM_WATCHPOINTS > 16
0b10	Select 32 to 47.	When NUM_BREAKPOINTS > 32 or NUM_WATCHPOINTS > 32
0b11	Select 48 to 63.	When NUM_BREAKPOINTS > 48 or NUM_WATCHPOINTS > 48

Each of the following register names accesses a register for breakpoint or watchpoint <n>, where n = UInt(MDSELR_EL1.BANK:m[3:0]):

- DBGBCR<m>_EL1.
- DBGBVR<m>_EL1.
- DBGWCR<m>_EL1.
- DBGWVR<m>_EL1.

This field is ignored by the PE and treated as zeros when any of the following are true:

- Executing at EL3 and MDCR_EL3.EBWE is 0.
- Executing at EL2 and the Effective value of MDCR_EL2.EBWE is 0.
- Executing at EL1 and the Effective value of MDSCR_EL1.EMBWE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES0** if all the following are true:
  - NUM_BREAKPOINTS <= 16.
  - NUM_WATCHPOINTS <= 16.
- Otherwise, access to this field is **RW**.

Bits [3:0]

Reserved, RES0.

Accessing MDSELR_EL1

When 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI, it is IMPLEMENTATION DEFINED whether these trap controls have any effect on accesses to MDSELR_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDSELR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_Debugv8p9) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EBWE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nMDSELR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EBWE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDSELR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EBWE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EBWE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDSELR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDSELR_EL1;
```

```
MSR MDSELR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_Debugv8p9) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EBWE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nMDSELR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EBWE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EBWE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EBWE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MDSELR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDSTEPOP_EL1, Monitor Debug Step Opcode Register

The MDSTEPOP_EL1 characteristics are:

Purpose

Used to execute instructions while Software Step is in active-not-pending state.

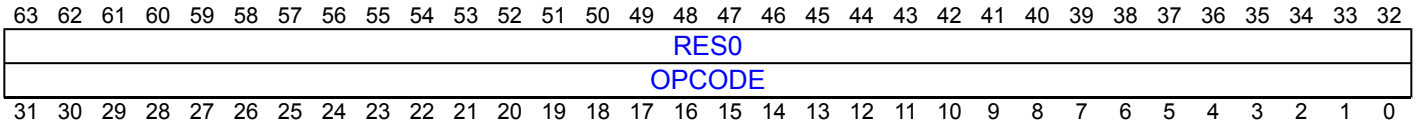
Configuration

This register is present only when FEAT_STEP2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MDSTEPOP_EL1 are UNDEFINED.

Attributes

MDSTEPOP_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

OPCODE, bits [31:0]

A64 instruction to be executed on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MDSTEPOP_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDSTEPOP_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0101	0b010

```
if !(IsFeatureImplemented(FEAT_STEP2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnSTEPOP == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nMDSTEPOP_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnSTEPOP == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDSTEPOP_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnSTEPOP == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnSTEPOP == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDSTEPOP_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDSTEPOP_EL1;
```

```
MSR MDSTEPOP_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0101	0b010

```
if !(IsFeatureImplemented(FEAT_STEP2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnSTEPOP == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nMDSTEPOP_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnSTEPOP == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSTEPOP_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnSTEPOP == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnSTEPOP == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSTEPOP_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MDSTEPOP_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MECID_A0_EL2, Alternate MECID for EL2 and EL2&0 translation regimes

The MECID_A0_EL2 characteristics are:

Purpose

Alternate MECID for EL2 and EL2&0 accesses translated by TTBR0_EL2.

Configuration

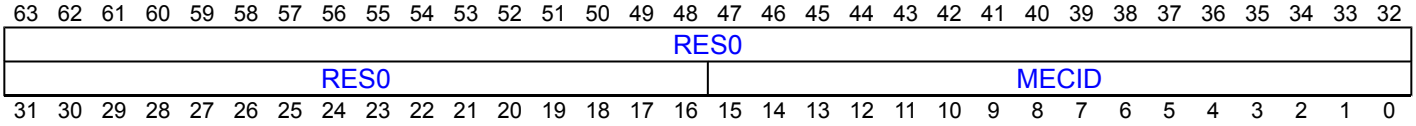
This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECID_A0_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

MECID_A0_EL2 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MECID_A0_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECID_A0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b001

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MECID_A0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECID_A0_EL2;
```

```
MSR MECID_A0_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b001

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MECID_A0_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MECID_A0_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MECID_A1_EL2, Alternate MECID for EL2&0 translation regimes.

The MECID_A1_EL2 characteristics are:

Purpose

Alternate MECID for EL2&0 accesses translated by TTBR1_EL2.

Configuration

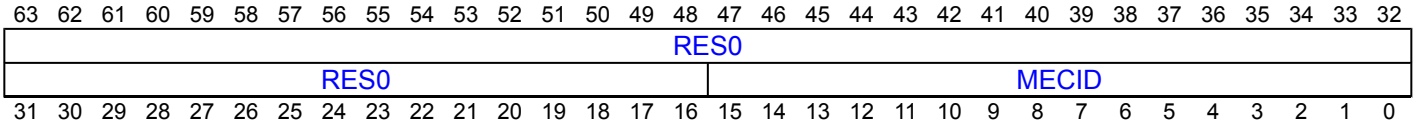
This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECID_A1_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

MECID_A1_EL2 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MECID_A1_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECID_A1_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MECID_A1_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECID_A1_EL2;
```

MSR MECID_A1_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MECID_A1_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MECID_A1_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MECID_P0_EL2, Primary MECID for EL2 and EL2&0 translation regimes

The MECID_P0_EL2 characteristics are:

Purpose

Primary MECID for EL2 and EL2&0 accesses translated by TTBR0_EL2.

Configuration

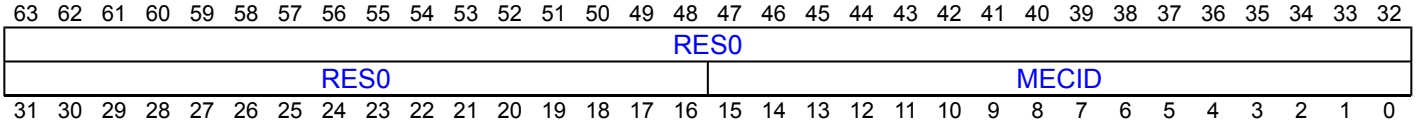
This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECID_P0_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

MECID_P0_EL2 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MECID_P0_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECID_P0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MECID_P0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECID_P0_EL2;
```

MSR MECID_P0_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MECID_P0_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MECID_P0_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MECID_P1_EL2, Primary MECID for EL2&0 translation regimes

The MECID_P1_EL2 characteristics are:

Purpose

Primary MECID for EL2&0 accesses translated by TTBR1_EL2.

Configuration

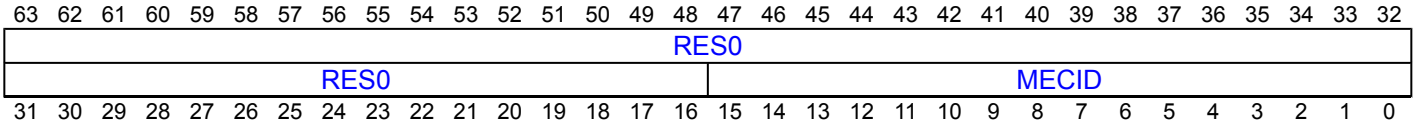
This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECID_P1_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

MECID_P1_EL2 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MECID_P1_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECID_P1_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b010

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MECID_P1_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECID_P1_EL2;
```

MSR MECID_P1_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b010

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MECID_P1_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MECID_P1_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MECID_RL_A_EL3, Realm PA space Alternate MECID for EL3 stage 1 translation regime

The MECID_RL_A_EL3 characteristics are:

Purpose

Alternate MECID for EL3 accesses to the Realm physical address space, translated by TTBR0_EL3.

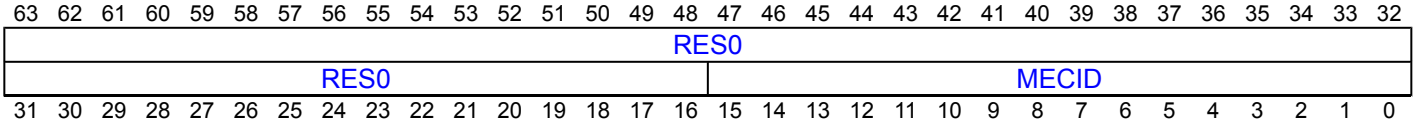
Configuration

This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECID_RL_A_EL3 are UNDEFINED.

Attributes

MECID_RL_A_EL3 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MECID_RL_A_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECID_RL_A_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b1010	0b001

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECID_RL_A_EL3;
```

MSR MECID_RL_A_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b1010	0b001

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.MECID_RL_A_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MECID_RL_A_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MECIDR_EL2, MEC Identification Register

The MECIDR_EL2 characteristics are:

Purpose

MEC identification register.

Configuration

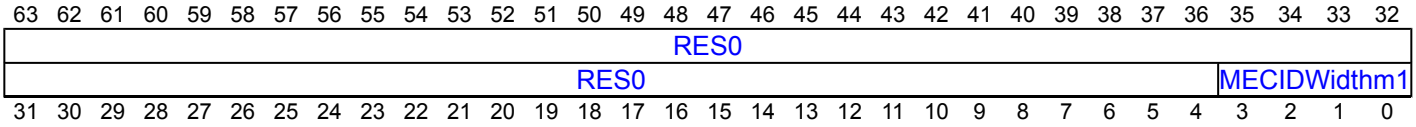
This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECIDR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

MECIDR_EL2 is a 64-bit register.

Field descriptions



Bits [63:4]

Reserved, RES0.

MECIDWidthm1, bits [3:0]

MECID width minus 1.

The value of this field plus 1 is the MECID width in bits, that this PE supports.

For example, the value `0b1111` indicates that this PE supports a MECID width of 16 bits, and provides 2^{16} possible MECID values.

MECIDWidth is defined as MECIDR_EL2.MECIDWidthm1 + 1.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MECIDR_EL2

For accesses from EL2 and EL3, this register is RO.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECIDR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b111

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MECIDR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECIDR_EL2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MFAR_EL3, Physical Fault Address Register (EL3)

The MFAR_EL3 characteristics are:

Purpose

Records the faulting physical address for a Granule Protection Check, synchronous External abort, or SError exception taken to EL3.

Configuration

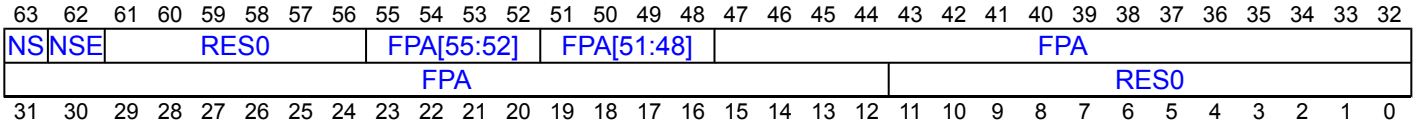
This register is present only when (FEAT_PFAR is implemented or FEAT_RME is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to MFAR_EL3 are UNDEFINED.

Attributes

MFAR_EL3 is a 64-bit register.

Field descriptions

When FEAT_RME is implemented and the exception is a GPC exception:



NS, bit [63]

Together with MFAR_EL3.NSE, reports the physical address space of the access that triggered the exception.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [62]

Together with MFAR_EL3.NS, reports the physical address space of the access that triggered the exception.

For a description of the values derived by evaluating NS and NSE together, see MFAR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [61:56]

Reserved, RES0.

FPA[55:52], bits [55:52]

When FEAT_D128 is implemented:

When FEAT_D128 is implemented, extension to MFAR_EL3.FPA[47:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FPA[51:48], bits [51:48]

When FEAT_LPA is implemented:

When FEAT_LPA is implemented, extension to MFAR_EL3.FPA[47:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FPA, bits [47:12]

Bits [47:12] of the Faulting Physical Address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

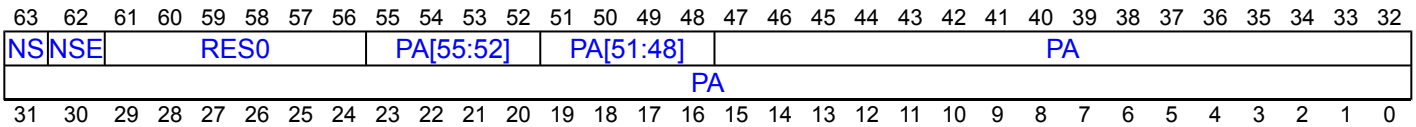
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

When FEAT_PFAR is implemented and the exception is a synchronous External abort or SError exception:



NS, bit [63]

When FEAT_RME is implemented:

Together with MFAR_EL3.NSE, reports the physical address space of the access that triggered the exception.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure. Reports the physical address space of the access that triggered the exception.

NS	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [62]

When FEAT_RME is implemented:

Together with MFAR_EL3.NS, reports the physical address space of the access that triggered the exception.

For a description of the values derived by evaluating NS and NSE together, see MFAR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [61:56]

Reserved, RES0.

PA[55:52], bits [55:52]

When FEAT_D128 is implemented:

When FEAT_D128 is implemented, extension to MFAR_EL3.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

When FEAT_LPA is implemented, extension to MFAR_EL3.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA, bits [47:0]

Physical Address. Bits [47:0] of the aborting physical address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The recorded address can be any address within the same naturally-aligned fault granule as the faulting physical address, where the size of the fault granule is IMPLEMENTATION DEFINED and no larger than the larger than:

- The size of the range of values permitted to be recorded in FAR_EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MFAR_EL3

MFAR_EL3 is not valid and reads UNKNOWN if ESR_EL3.EC is recorded indicating an Abort or SError exception and ESR_EL3.PFV is recorded as 0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MFAR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0110	0b0000	0b101

```
if !((IsFeatureImplemented(FEAT_PFAR) || IsFeatureImplemented(FEAT_RME)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MFAR_EL3;
```

```
MSR MFAR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0110	0b0000	0b101

```
if !((IsFeatureImplemented(FEAT_PFAR) || IsFeatureImplemented(FEAT_RME)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MFAR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MIDR_EL1, Main ID Register

The MIDR_EL1 characteristics are:

Purpose

Provides identification information for the PE, including an implementer code for the device and a device ID number.

Configuration

AArch64 System register MIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MIDR[31:0].

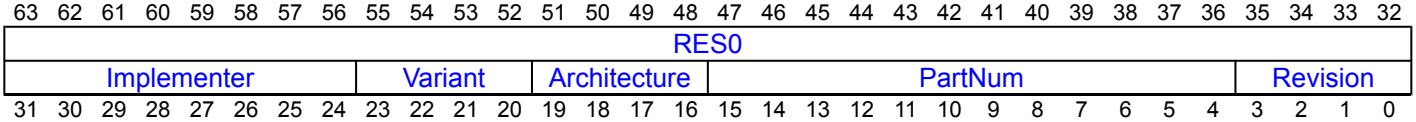
AArch64 System register MIDR_EL1 bits [31:0] are architecturally mapped to External register MIDR_EL1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MIDR_EL1 are UNDEFINED.

Attributes

MIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Implementer	Meaning
0x00	Reserved for software use.
0x41	Arm Limited.
0x42	Broadcom Corporation.
0x43	Cavium Inc.
0x44	Digital Equipment Corporation.
0x46	Fujitsu Ltd.
0x49	Infineon Technologies AG.
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation.
0x50	Applied Micro Circuits Corporation.
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation.
0xC0	Ampere Computing.

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

Access to this field is **RO**.

Variant, bits [23:20]

Variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Architecture, bits [19:16]

Architecture version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Architecture	Meaning
0b0001	Armv4.
0b0010	Armv4T.
0b0011	Armv5 (obsolete).
0b0100	Armv5T.
0b0101	Armv5TE.
0b0110	Armv5TEJ.
0b0111	Armv6.
0b1111	Architectural features are individually identified in the ID_* registers.

All other values are reserved.

Access to this field is **RO**.

PartNum, bits [15:4]

Primary Part Number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [3:0]

Revision number for the device.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.MIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() then
        X[t, 64] = VPIDR_EL2;
    else
        X[t, 64] = MIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAM0_EL1, MPAM0 Register (EL1)

The MPAM0_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL0. When EL2 is implemented and enabled in the current Security state, the MPAM virtualization option is present, MPAMHCR_EL2.GSTAPP_PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM information to label memory requests.

If EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the MPAM virtualization option is present and MPAMHCR_EL2.EL0_VPMEN == 1, then MPAM PARTIDs in MPAM0_EL1 are virtual and mapped into physical PARTIDs for the current Security state.

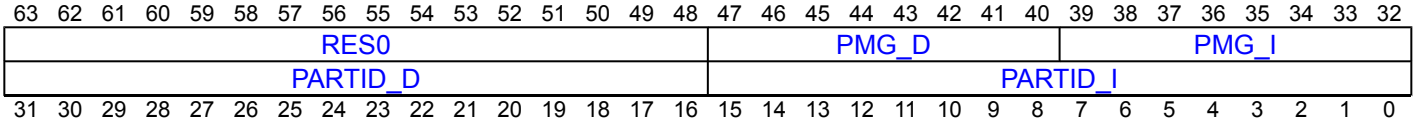
Configuration

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM0_EL1 are UNDEFINED.

Attributes

MPAM0_EL1 is a 64-bit register.

Field descriptions



Bits [63:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM0_EL1

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAM0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM0EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAM0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAM0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAM0_EL1;
```

```
MSR MPAM0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM0EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        MPAM0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAM0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAM0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAM1_EL1, MPAM1 Register (EL1)

The MPAM1_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL1.

When EL2 is implemented and enabled in the current Security state, the MPAM virtualization option is present, MPAMHCR_EL2.GSTAPP_PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM labels for memory requests when executing at EL0.

If EL2 is implemented and enabled in the current Security state, the MPAM virtualization option is present and MPAMHCR_EL2.EL1_VPMEN == 1, MPAM PARTIDs in MPAM1_EL1 are virtual and mapped into physical PARTIDs for the current Security state. This mapping of MPAM1_EL1 virtual PARTIDs to physical PARTIDs when EL1_VPMEN is 1 also applies when MPAM1_EL1 is used at EL0 due to MPAMHCR_EL2.GSTAPP_PLK.

Configuration

AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register MPAM3_EL3[63] when EL3 is implemented.

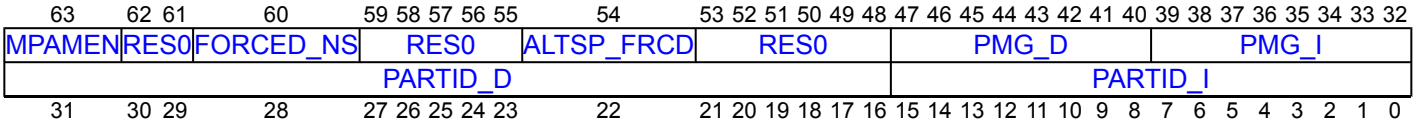
AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register MPAM2_EL2[63] when EL3 is not implemented and EL2 is implemented.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM1_EL1 are UNDEFINED.

Attributes

MPAM1_EL1 is a 64-bit register.

Field descriptions



MPAMEN, bit [63]

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

MPAMEN	Meaning
0b0	The default PARTID and default PMG are output in MPAM information.
0b1	MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.

If EL3 is implemented, this field is read-only and reads the current value of the read/write bit MPAM3_EL3.MPAMEN.

If EL3 is not implemented and EL2 is implemented, this field is read-only and reads the current value of the read/write bit MPAM2_EL2.MPAMEN.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RW** if all the following are true:
  - EL3 is not implemented.
  - EL2 is not implemented.

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - MPAM3_EL3.MPAMEN == 0.
- Access to this field is **RAO/WI** if all the following are true:
  - EL3 is implemented.
  - MPAM3_EL3.MPAMEN == 1.
- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is not implemented.
  - EL2 is implemented.
  - MPAM2_EL2.MPAMEN == 0.
- Access to this field is **RAO/WI** if all the following are true:
  - EL3 is not implemented.
  - EL2 is implemented.
  - MPAM2_EL2.MPAMEN == 1.

Bits [62:61]

Reserved, RES0.

FORCED_NS, bit [60]

When FEAT_MPAMv0p1 is implemented:

In the Secure state, FORCED_NS indicates the state of MPAM3_EL3.FORCE_NS.

FORCED_NS	Meaning
0b0	In the Non-secure state, always reads as 0. In the Secure state, indicates that MPAM3_EL3.FORCE_NS == 0.
0b1	In the Secure state, indicates that MPAM3_EL3.FORCE_NS == 1.

Always reads as 0 in the Non-secure state.

Writes are ignored.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [59:55]

Reserved, RES0.

ALTSP_FRCD, bit [54]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Alternative PARTID forced for PARTIDs in this register.

ALTSP_FRCD	Meaning
0b0	The PARTIDs in MPAM1_EL1 and MPAM0_EL1 are using the primary PARTID space.
0b1	The PARTIDs in MPAM1_EL1 and MPAM0_EL1 are using the alternative PARTID space.

This bit indicates that a higher Exception level has forced the PARTIDs in this register to use the alternative PARTID space defined for the current Security state.

In MPAM1_EL1, it also indicates that MPAM0_EL1 is forced to use alternative PARTID space.

For more information, see 'In a PE with FEAT_RME, selection of primary or alternative PARTID space'.

Accessing this field has the following behavior:

- When !UsePrimarySpaceEL10(), access to this field is **RAO/WI**.
- Otherwise, access to this field is **RAZ/WI**.

Otherwise:

Reserved, RES0.

Bits [53:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM1_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name MPAM1_EL1 or MPAM1_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAM1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x900];
    else
        X[t, 64] = MPAM1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = MPAM2_EL2;
    else
        X[t, 64] = MPAM1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAM1_EL1;
```

```
MSR MPAM1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x900] = X[t, 64];
    else
        MPAM1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MPAM2_EL2 = X[t, 64];
    else
        MPAM1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAM1_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, MPAM1_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x900];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = MPAM1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = MPAM1_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR MPAM1_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x900] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAM1_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        MPAM1_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAM2_EL2, MPAM2 Register (EL2)

The MPAM2_EL2 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL2.

Configuration

AArch64 System register MPAM2_EL2 bit [63] is architecturally mapped to AArch64 System register MPAM3_EL3[63] when EL3 is implemented.

AArch64 System register MPAM2_EL2 bit [63] is architecturally mapped to AArch64 System register MPAM1_EL1[63].

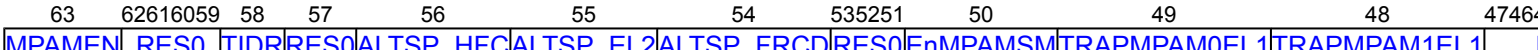
This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM2_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAM2_EL2 is a 64-bit register.

Field descriptions



MPAMEN, bit [63]

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

MPAMEN	Meaning
0b0	The default PARTID and default PMG are output in MPAM information from all Exception levels.
0b1	MPAM information is output based on the MPAMn_ELx register for ELn according to the MPAM configuration.

If EL3 is implemented, this field is read-only and reads the current value of the read/write MPAM3_EL3.MPAMEN bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - MPAM3_EL3.MPAMEN == 0.
- Access to this field is **RAO/WI** if all the following are true:
  - EL3 is implemented.
  - MPAM3_EL3.MPAMEN == 1.
- When EL3 is not implemented, access to this field is **RW**.

Bits [62:59]

Reserved, RES0.

TIDR, bit [58]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMIDR_EL1.HAS_TIDR == 1:

TIDR traps accesses to MPAMIDR_EL1 from EL1 to EL2.

TIDR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Trap accesses to MPAMIDR_EL1 from EL1 to EL2.

MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == 1 also traps MPAMIDR_EL1 accesses from EL1 to EL2. If either TIDR or TRAP_MPAMIDR_EL1 are 1, accesses are trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [57]

Reserved, RES0.

ALTSP_HFC, bit [56]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Hierarchical force of alternative PARTID space controls. When MPAM3_EL3.ALTSP_HEN is 0, ALTSP controls in MPAM2_EL2 have no effect. When MPAM3_EL3.ALTSP_HEN is 1, this bit selects whether the PARTIDs in MPAM1_EL1 and MPAM0_EL1 are in the primary (0) or alternative (1) PARTID space for the security state.

ALTSP_HFC	Meaning
0b0	When MPAM3_EL3.ALTSP_HEN is 1, the PARTID space of MPAM1_EL1.PARTID_I, MPAM1_EL1.PARTID_D, MPAM0_EL1.PARTID_I, and MPAM0_EL1.PARTID_D are in the primary PARTID space for the Security state.
0b1	When MPAM3_EL3.ALTSP_HEN is 1, the PARTID space of MPAM1_EL1.PARTID_I, MPAM1_EL1.PARTID_D, MPAM0_EL1.PARTID_I, and MPAM0_EL1.PARTID_D are in the alternative PARTID space for the Security state.

This control has no effect when MPAM3_EL3.ALTSP_HEN is 0.

For more information, see 'In a PE with FEAT_RME, selection of primary or alternative PARTID space'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_EL2, bit [55]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Select alternative PARTID space for PARTIDs in MPAM2_EL2 when MPAM3_EL3.ALTSP_HEN is 1.

ALTSP_EL2	Meaning
0b0	When MPAM3_EL3.ALTSP_HEN is 1, selects the primary PARTID space for MPAM2_EL2.PARTID_I and MPAM2_EL2.PARTID_D.
0b1	When MPAM3_EL3.ALTSP_HEN is 1, selects the alternative PARTID space for MPAM2_EL2.PARTID_I and MPAM2_EL2.PARTID_D.

For more information, see 'In a PE with FEAT_RME, selection of primary or alternative PARTID space'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_FRCD, bit [54]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Alternative PARTID forced for PARTIDs in this register.

ALTSP_FRCD	Meaning
0b0	The PARTIDs in this register are using the primary PARTID space.
0b1	The PARTIDs in this register are using the alternative PARTID space.

This bit indicates that a higher Exception level has forced the PARTIDs in this register to use the alternative PARTID space defined for the current Security state. In EL2, it is also 1 when MPAM2_EL2.ALTSP_EL2 is 1.

For more information, see 'In a PE with FEAT_RME, selection of primary or alternative PARTID space'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When !UsePrimarySpaceEL2(), access to this field is **RAO/WI**.
- Otherwise, access to this field is **RAZ/WI**.

Otherwise:

Reserved, RES0.

Bits [53:51]

Reserved, RES0.

EnMPAMSM, bit [50]

When FEAT_SME is implemented:

Traps execution at EL1 of instructions that directly access the MPAMSM_EL1 register to EL2. The exception is reported using ESR_ELx.EC syndrome value 0x18.

EnMPAMSM	Meaning
0b0	This control causes execution of these instructions at EL1 to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

This field has no effect on accesses to MPAMSM_EL1 from EL2 or EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRAPMPAM0EL1, bit [49]

Trap accesses from EL1 to the MPAM0_EL1 register trap to EL2.

TRAPMPAM0EL1	Meaning
0b0	Accesses to MPAM0_EL1 from EL1 are not trapped.
0b1	Accesses to MPAM0_EL1 from EL1 are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When EL3 is not implemented, this field resets to '1'.
  - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

TRAPMPAM1EL1, bit [48]

Trap accesses from EL1 to the MPAM1_EL1 register trap to EL2.

TRAPMPAM1EL1	Meaning
0b0	Accesses to MPAM1_EL1 from EL1 are not trapped.
0b1	Accesses to MPAM1_EL1 from EL1 are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When EL3 is not implemented, this field resets to '1'.
  - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name MPAM2_EL2 or MPAM1_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAM2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAM2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAM2_EL2;
```

```
MSR MPAM2_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAM2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAM2_EL2 = X[t, 64];
```

MRS <Xt>, MPAM1_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x900];
    else
        X[t, 64] = MPAM1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = MPAM2_EL2;
    else
        X[t, 64] = MPAM1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAM1_EL1;
```

```
MSR MPAM1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x900] = X[t, 64];
    else
        MPAM1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MPAM2_EL2 = X[t, 64];
    else
        MPAM1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAM1_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAM3_EL3, MPAM3 Register (EL3)

The MPAM3_EL3 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL3.

Configuration

AArch64 System register MPAM3_EL3 bit [63] is architecturally mapped to AArch64 System register MPAM2_EL2[63] when EL2 is implemented.

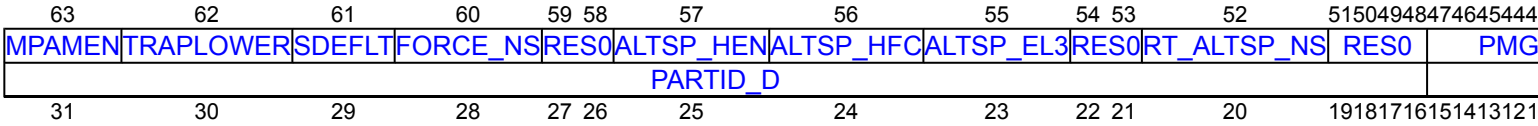
AArch64 System register MPAM3_EL3 bit [63] is architecturally mapped to AArch64 System register MPAM1_EL1[63].

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM3_EL3 are UNDEFINED.

Attributes

MPAM3_EL3 is a 64-bit register.

Field descriptions



MPAMEN, bit [63]

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

Values of this field are:

MPAMEN	Meaning
0b0	The default PARTID and default PMG are output in MPAM information when executing at any ELn.
0b1	MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **RW**.

TRAPLOWER, bit [62]

Trap direct accesses to MPAM System registers that are not UNDEFINED from all ELn lower than EL3.

TRAPLOWER	Meaning
0b0	Do not force trapping of direct accesses of MPAM System registers to EL3.
0b1	Force direct accesses of MPAM System registers to trap to EL3.

Note

This trap is higher priority than any of the traps controlled by MPAM _EL2 registers.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'1'`.

SDEFLT, bit [61]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMIDR_EL1.HAS_SDEFLT == 1:

SDEFLT overrides the PARTID and PMG with the default PARTID and default PMG when executing in the Secure state.

SDEFLT	Meaning
`0b0`	The PARTID and PMG are determined normally in the Secure state.
`0b1`	When executing in the Secure state, the PARTID is always PARTID 0, and the PMG is always PMG 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FORCE_NS, bit [60]

When FEAT_MPAMv0p1 is implemented and MPAMIDR_EL1.HAS_FORCE_NS == 1:

FORCE_NS forces MPAM_NS to always be 1 in the Secure state.

FORCE_NS	Meaning
`0b0`	MPAM_NS is 0 when executing in the Secure state.
`0b1`	MPAM_NS is 1 when executing in the Secure state.

An implementation is permitted to have this field as RAO if the implementation does not support generating MPAM_NS as 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [59:58]

Reserved, RES0.

ALTSP_HEN, bit [57]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Hierarchical enable for alternative PARTID space controls. Alternative PARTID space controls in MPAM2_EL2 have no effect when this field is zero.

ALTSP_HEN	Meaning
`0b0`	Disable alternative PARTID space controls in MPAM2_EL2. The PARTID space for PARTIDs in MPAM2_EL2, MPAM1_EL1, and MPAM0_EL1 is selected by MPAM3_EL3.ALTSP_HFC.
`0b1`	Enable alternative PARTID space controls in MPAM2_EL2 to control the PARTID space used for PARTIDs in MPAM2_EL2, MPAM1_EL1, and MPAM0_EL1.

For more information, see 'In a PE with FEAT_RME, selection of primary or alternative PARTID space'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_HFC, bit [56]
When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Hierarchical force of alternative PARTID space controls. When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space for PARTIDs in MPAM2_EL2, MPAM1_EL1, and MPAM0_EL1 is selected by the value of this bit.

ALTSP_HFC	Meaning
0b0	When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space of MPAM2_EL2.PARTID, MPAM1_EL1.PARTID and MPAM0_EL1.PARTID are the primary PARTID space for the security state.
0b1	When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space of MPAM2_EL2.PARTID and MPAM1_EL1.PARTID and MPAM0_EL1.PARTID are the alternative PARTID space for the security state.

For more information, see 'In a PE with FEAT_RME, selection of primary or alternative PARTID space'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_EL3, bit [55]
When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Select alternative PARTID space for PARTIDs in MPAM3_EL3.

ALTSP_EL3	Meaning
0b0	Selects the primary PARTID space of MPAM3_EL3.PARTID_I and MPAM3_EL3.PARTID_D.
0b1	Selects the alternative PARTID space of MPAM3_EL3.PARTID_I and MPAM3_EL3.PARTID_D.

For more information, see 'In a PE with FEAT_RME, selection of primary or alternative PARTID space'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

Bits [54:53]

Reserved, RES0.

RT_ALTSP_NS, bit [52]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Selects whether the alternative PARTID space for the Root security state is the Secure PARTID space or the Non-secure PARTID space. MPAM3_EL3.RT_ALTSP_NS selects the alternative PARTID space for the Root Security state when MPAM3_EL3.ALTSP_EL3 == 1.

RT_ALTSP_NS	Meaning
0b0	The alternative PARTID space in the Root security state is the Secure PARTID space.
0b1	The alternative PARTID space in the Root security state is the Non-secure PARTID space.

This field has no effect except in the Root security state (EL3).

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

Bits [51:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM3_EL3

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAM3_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAM3_EL3;
```

MSR MPAM3_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.MPAM3_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAM3_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMBW0_EL1, MPAM PE-side Maximum-bandwidth Control Register (EL0)

The MPAMBW0_EL1 characteristics are:

Purpose

Enables software to configure a maximum fraction of memory bandwidth that the PE is permitted to use when executing at EL0 with its current PARTID.

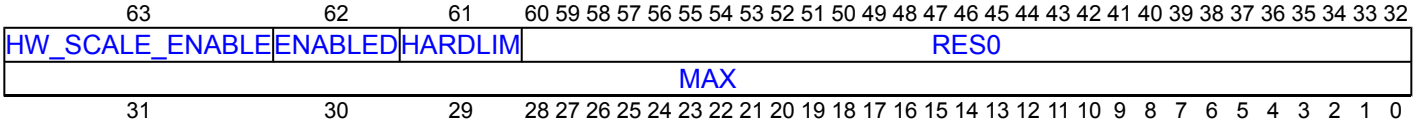
Configuration

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented. Otherwise, direct accesses to MPAMBW0_EL1 are UNDEFINED.

Attributes

MPAMBW0_EL1 is a 64-bit register.

Field descriptions



HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == 1:

Enables hardware bandwidth scaling of the MPAMBW0_EL1.MAX value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling in EL0 is disabled.
0b1	PE-side memory bandwidth control hardware scaling in EL0 is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control when in EL0.

ENABLED	Meaning
0b0	The PE-side memory bandwidth control in EL0 is disabled.
0b1	The PE-side memory bandwidth control in EL0 is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HARDLIM, bit [61]

PE-side Maximum Bandwidth Limit Behavior Selection.

HARDLIM	Meaning
0b0	Soft limit: when MPAMBW0_EL1.MAX bandwidth is exceeded, the PE is unregulated unless the downstream memory path is saturated. It is IMPLEMENTATION DEFINED how hardware determines when the downstream memory path is saturated.
0b1	Hard limit: when MPAMBW0_EL1.MAX bandwidth is exceeded, the PE does not use any more bandwidth until the memory bandwidth for the PE falls below MPAMBW0_EL1.MAX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

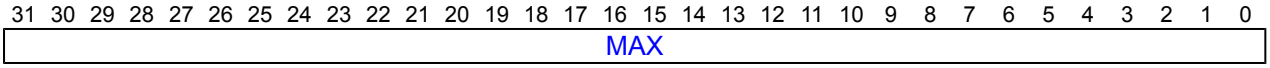
- When MPAMBWIDR_EL1.MAX_LIM == 0b00, access to this field is **RW**.
- When MPAMBWIDR_EL1.MAX_LIM == 0b01, access to this field is **RAZ/WI**.
- When MPAMBWIDR_EL1.MAX_LIM == 0b10, access to this field is **RAO/WI**.

Bits [60:32]

Reserved, RES0.

MAX, bits [31:0]

MAX encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 1 and MPAMBW0_EL1.HW_SCALE_ENABLE == 1



MAX, bits [31:0]

Maximum memory bandwidth allocated to the PE when executing at EL0 with its current PARTID.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

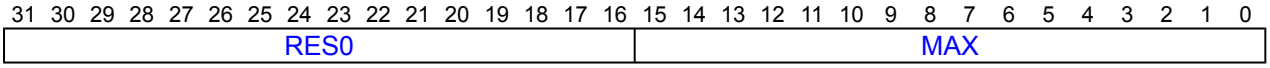
Bits [31:16] represent the integer part of the value.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MAX encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 0 or MPAMBW0_EL1.HW_SCALE_ENABLE == 0



Bits [31:16]

Reserved, RES0.

MAX, bits [15:0]

Maximum memory bandwidth allocated to the PE when executing at EL0 with its current PARTID.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBW0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBW0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b101

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAMBW0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMBW0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBW0_EL1;
```

```
MSR MPAMBW0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b101

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        MPAMBW0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMBW0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBW0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMBW1_EL1, MPAM PE-side Maximum-bandwidth Control Register (EL1)

The MPAMBW1_EL1 characteristics are:

Purpose

Enables software to configure a maximum fraction of memory bandwidth that the PE is permitted to use when executing at EL1 with its current PARTID.

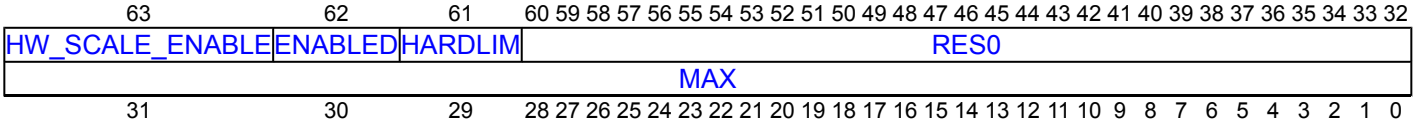
Configuration

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented. Otherwise, direct accesses to MPAMBW1_EL1 are UNDEFINED.

Attributes

MPAMBW1_EL1 is a 64-bit register.

Field descriptions



HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == 1:

Enables hardware bandwidth scaling of the MPAMBW1_EL1.MAX value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling in EL1 is disabled.
0b1	PE-side memory bandwidth control hardware scaling in EL1 is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control when in EL1.

ENABLED	Meaning
0b0	The PE-side memory bandwidth control in EL1 is disabled.
0b1	The PE-side memory bandwidth control in EL1 is enabled.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL1, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

HARDLIM, bit [61]

PE-side Maximum Bandwidth Limit Behavior Selection.

HARDLIM	Meaning
0b0	Soft limit: when MPAMBW1_EL1.MAX bandwidth is exceeded, the PE is unregulated unless the downstream memory path is saturated. It is IMPLEMENTATION DEFINED how hardware determines when the downstream memory path is saturated.
0b1	Hard limit: when MPAMBW1_EL1.MAX bandwidth is exceeded, the PE does not use any more bandwidth until the memory bandwidth for the PE falls below MPAMBW1_EL1.MAX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

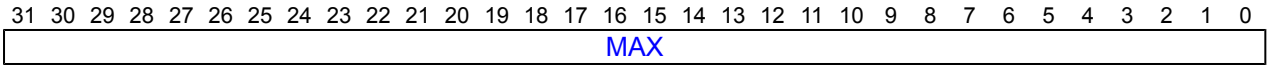
- When MPAMBWIDR_EL1.MAX_LIM == 0b00, access to this field is **RW**.
- When MPAMBWIDR_EL1.MAX_LIM == 0b01, access to this field is **RAZ/WI**.
- When MPAMBWIDR_EL1.MAX_LIM == 0b10, access to this field is **RAO/WI**.

Bits [60:32]

Reserved, RES0.

MAX, bits [31:0]

MAX encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 1 and MPAMBW1_EL1.HW_SCALE_ENABLE == 1



MAX, bits [31:0]

Maximum memory bandwidth allocated to the PE when executing at EL1 with its current PARTID.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

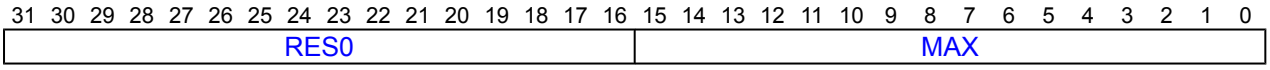
Bits [31:16] represent the integer part of the value.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MAX encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 0 or MPAMBW1_EL1.HW_SCALE_ENABLE == 0



Bits [31:16]

Reserved, RES0.

MAX, bits [15:0]

Maximum memory bandwidth allocated to the PE when executing at EL1 with its current PARTID.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBW1_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name MPAMBW1_EL1 or MPAMBW1_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBW1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW1_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x908];
    else
        X[t, 64] = MPAMBW1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = MPAMBW2_EL2;
    else
        X[t, 64] = MPAMBW1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBW1_EL1;
```

```
MSR MPAMBW1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW1_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x908] = X[t, 64];
    else
        MPAMBW1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MPAMBW2_EL2 = X[t, 64];
    else
        MPAMBW1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBW1_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, MPAMBW1_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x908];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = MPAMBW1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = MPAMBW1_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR MPAMBW1_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x908] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMBW1_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        MPAMBW1_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMBW2_EL2, MPAM PE-side Maximum-bandwidth Control Register (EL2)

The MPAMBW2_EL2 characteristics are:

Purpose

Enables software to configure a maximum fraction of memory bandwidth that the PE is permitted to use when executing at EL2 with its current PARTID.

Configuration

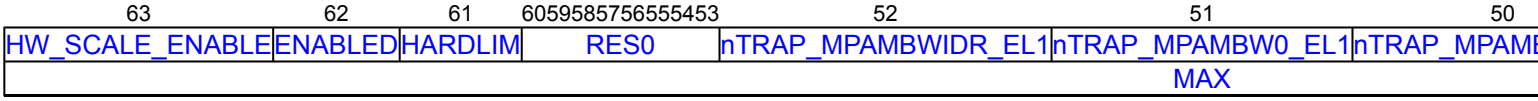
This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented. Otherwise, direct accesses to MPAMBW2_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMBW2_EL2 is a 64-bit register.

Field descriptions



HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == 1:

Enables hardware bandwidth scaling of the MPAMBW2_EL2.MAX value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling in EL2 is disabled.
0b1	PE-side memory bandwidth control hardware scaling in EL2 is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control when in EL2.

ENABLED	Meaning
0b0	The PE-side memory bandwidth control in EL2 is disabled.
0b1	The PE-side memory bandwidth control in EL2 is enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

HARDLIM, bit [61]

PE-side Maximum Bandwidth Limit Behavior Selection.

HARDLIM	Meaning
0b0	Soft limit: when MPAMBW2_EL2.MAX bandwidth is exceeded, the PE is unregulated unless the downstream memory path is saturated. It is IMPLEMENTATION DEFINED how hardware determines when the downstream memory path is saturated.
0b1	Hard limit: when MPAMBW2_EL2.MAX bandwidth is exceeded, the PE does not use any more bandwidth until the memory bandwidth for the PE falls below MPAMBW2_EL2.MAX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MPAMBWIDR_EL1.MAX_LIM == 0b00, access to this field is **RW**.
- When MPAMBWIDR_EL1.MAX_LIM == 0b01, access to this field is **RAZ/WI**.
- When MPAMBWIDR_EL1.MAX_LIM == 0b10, access to this field is **RAO/WI**.

Bits [60:53]

Reserved, RES0.

nTRAP_MPAMBWIDR_EL1, bit [52]

Traps accesses to MPAMBWIDR_EL1 from EL1 to EL2.

nTRAP_MPAMBWIDR_EL1	Meaning
0b0	Accesses to MPAMBWIDR_EL1 from EL1 are trapped to EL2 with EC syndrome value 0x18.
0b1	Accesses to MPAMBWIDR_EL1 from EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

nTRAP_MPAMBW0_EL1, bit [51]

Traps accesses to MPAMBW0_EL1 from EL1 to EL2.

nTRAP_MPAMBW0_EL1	Meaning
0b0	Accesses to MPAMBW0_EL1 from EL1 are trapped to EL2 with EC syndrome value 0x18.
0b1	Accesses to MPAMBW0_EL1 from EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

nTRAP_MPAMBW1_EL1, bit [50]

Traps accesses to MPAMBW1_EL1 from EL1 to EL2.

nTRAP_MPAMBW1_EL1	Meaning
0b0	Accesses to MPAMBW1_EL1 from EL1 are trapped to EL2 with EC syndrome value 0x18.
0b1	Accesses to MPAMBW1_EL1 from EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

nTRAP_MPAMBWSM_EL1, bit [49]

When FEAT_SME is implemented:

Traps accesses to MPAMBWSM_EL1 from EL1 to EL2.

nTRAP_MPAMBWSM_EL1	Meaning
0b0	Accesses to MPAMBWSM_EL1 from EL1 are trapped to EL2 with EC syndrome value 0x18.
0b1	Accesses to MPAMBWSM_EL1 from EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

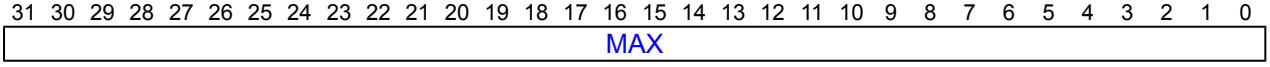
Reserved, RES0.

Bits [48:32]

Reserved, RES0.

MAX, bits [31:0]

MAX encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 1 and MPAMBW2_EL2.HW_SCALE_ENABLE == 1



MAX, bits [31:0]

Maximum memory bandwidth allocated to the PE when executing at EL2 with its current PARTID.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

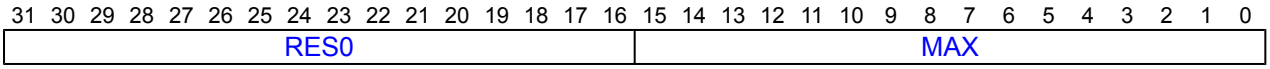
Bits [31:16] represent the integer part of the value.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MAX encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 0 or MPAMBW2_EL2.HW_SCALE_ENABLE == 0



Bits [31:16]

Reserved, RES0.

MAX, bits [15:0]

Maximum memory bandwidth allocated to the PE when executing at EL2 with its current PARTID.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBW2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name MPAMBW2_EL2 or MPAMBW1_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBW2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMBW2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBW2_EL2;
```

```
MSR MPAMBW2_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMBW2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBW2_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, MPAMBW1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW1_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x908];
    else
        X[t, 64] = MPAMBW1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = MPAMBW2_EL2;
    else
        X[t, 64] = MPAMBW1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBW1_EL1;
```

When FEAT_VHE is implemented

```
MSR MPAMBW1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW1_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x908] = X[t, 64];
    else
        MPAMBW1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MPAMBW2_EL2 = X[t, 64];
    else
        MPAMBW1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBW1_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMBW3_EL3, MPAM PE-side Maximum-bandwidth Control Register (EL3)

The MPAMBW3_EL3 characteristics are:

Purpose

Enables software to configure a maximum fraction of memory bandwidth that the PE is permitted to use when executing at EL3 with its current PARTID.

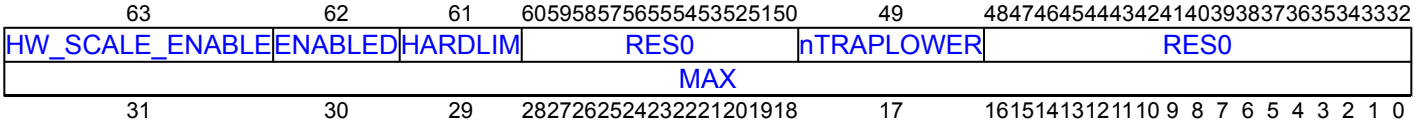
Configuration

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented. Otherwise, direct accesses to MPAMBW3_EL3 are UNDEFINED.

Attributes

MPAMBW3_EL3 is a 64-bit register.

Field descriptions



HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == 1:

Enables hardware bandwidth scaling of the MPAMBW3_EL3.MAX value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling in EL3 is disabled.
0b1	PE-side memory bandwidth control hardware scaling in EL3 is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control when in EL3.

ENABLED	Meaning
0b0	The PE-side memory bandwidth control in EL3 is disabled.
0b1	The PE-side memory bandwidth control in EL3 is enabled.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

HARDLIM, bit [61]

PE-side Maximum Bandwidth Limit Behavior Selection.

HARDLIM	Meaning
0b0	Soft limit: when MPAMBW3_EL3.MAX bandwidth is exceeded, the PE is unregulated unless the downstream memory path is saturated. It is IMPLEMENTATION DEFINED how hardware determines when the downstream memory path is saturated.
0b1	Hard limit: when MPAMBW3_EL3.MAX bandwidth is exceeded, the PE does not use any more bandwidth until the memory bandwidth for the PE falls below MPAMBW3_EL3.MAX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MPAMBWIDR_EL1.MAX_LIM == 0b00, access to this field is **RW**.
- When MPAMBWIDR_EL1.MAX_LIM == 0b01, access to this field is **RAZ/WI**.
- When MPAMBWIDR_EL1.MAX_LIM == 0b10, access to this field is **RAO/WI**.

Bits [60:50]

Reserved, RES0.

nTRAPLOWER, bit [49]

Traps accesses to MPAMBW2_EL2, MPAMBWCAP_EL2, MPAMBW1_EL1, MPAMBW0_EL1, MPAMBWSM_EL1, MPAMBWIDR_EL1 from any lower EL to EL3.

nTRAPLOWER	Meaning
0b0	Accesses to MPAMBW2_EL2, MPAMBWCAP_EL2, MPAMBW1_EL1, MPAMBW0_EL1, MPAMBWSM_EL1, MPAMBWIDR_EL1 from EL1 are trapped to EL3 with EC syndrome value 0x18.
0b1	Accesses to MPAMBW2_EL2, MPAMBWCAP_EL2, MPAMBW1_EL1, MPAMBW0_EL1, MPAMBWSM_EL1, MPAMBWIDR_EL1 from EL1 are not trapped by this mechanism.

Note

This trap is higher priority than any of the traps controlled by MPAM _EL2 registers.

The reset behavior of this field is:

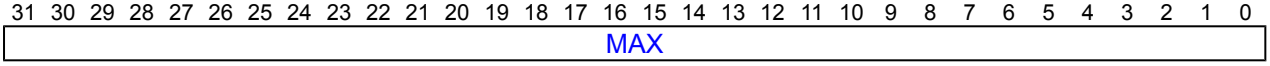
- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Bits [48:32]

Reserved, RES0.

MAX, bits [31:0]

MAX encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 1 and MPAMBW3_EL3.HW_SCALE_ENABLE == 1



MAX, bits [31:0]

Maximum memory bandwidth allocated to the PE when executing at EL3 with its current PARTID.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

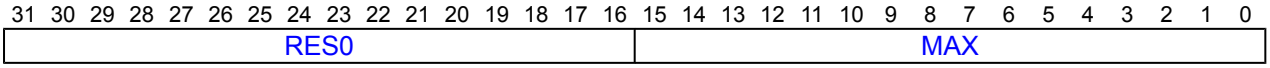
Bits [31:16] represent the integer part of the value.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MAX encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 0 or MPAMBW3_EL3.HW_SCALE_ENABLE == 0



Bits [31:16]

Reserved, RES0.

MAX, bits [15:0]

Maximum memory bandwidth allocated to the PE when executing at EL3 with its current PARTID.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBW3_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBW3_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBW3_EL3;
```

```
MSR MPAMBW3_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MPAMBW3_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMBWCAP_EL2, MPAM PE-side Maximum-bandwidth Limit Virtualization Register

The MPAMBWCAP_EL2 characteristics are:

Purpose

Allows software executing at EL2 to provide MPAMBWCAP_EL2.CAP as an upper bound to MPAMBW1_EL1.MAX.

If FEAT_SME is implemented, the upper bound also applies to MPAMBWSM_EL1.MAX when executing at EL1: the maximum bandwidth allowed for the PE is MIN(MPAMBWSM_EL1.MAX, MPAMBWCAP_EL2.CAP).

If the Effective value of HCR_EL2.{E2H,TGE} is not {1,1}:

- The upper bound also applies to MPAMBW0_EL1.MAX: the maximum bandwidth allowed for the PE is MIN(MPAMBW0_EL1.MAX, MPAMBWCAP_EL2.CAP).
- If FEAT_SME is implemented, the upper bound also applies to MPAMBWSM_EL1.MAX when executing at EL0: the maximum bandwidth allowed for the PE is MIN(MPAMBWSM_EL1.MAX, MPAMBWCAP_EL2.CAP).

If MPAMBWCAP_EL2.ENABLED is 1, a PARTID that has used more than min(CAP,MAX) is given no access to additional bandwidth.

Configuration

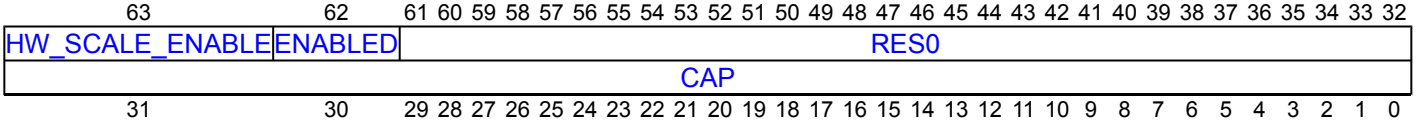
This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMBWCAP_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMBWCAP_EL2 is a 64-bit register.

Field descriptions



HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == 1:

Enables hardware bandwidth scaling of the MPAMBWCAP_EL2.CAP value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling for EL2 capping is disabled.
0b1	PE-side memory bandwidth control hardware scaling for EL2 capping is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control capping by EL2.

ENABLED	Meaning
0b0	The PE-side memory bandwidth control capping by EL2 is disabled.
0b1	The PE-side memory bandwidth control capping by EL2 is enabled.

The reset behavior of this field is:

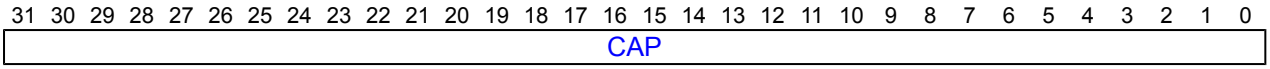
- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [61:32]

Reserved, RES0.

CAP, bits [31:0]

CAP encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 1 and MPAMBWCAP_EL2.HW_SCALE_ENABLE == 1



CAP, bits [31:0]

Upper bound to the maximum memory bandwidth allocated to the current PARTID in MPAMBW1_EL1.MAX, MPAMBW0_EL1.MAX and MPAMBWSM_EL1.MAX.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [31:16] represent the integer part of the value.

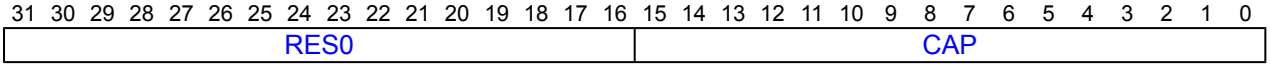
Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The value set in the MAX field must be less than or equal to this upper bound.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CAP encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 0 or MPAMBWCAP_EL2.HW_SCALE_ENABLE == 0



Bits [31:16]

Reserved, RES0.

CAP, bits [15:0]

Upper bound to the maximum memory bandwidth allocated to the current PARTID in MPAMBW1_EL1.MAX, MPAMBW0_EL1.MAX and MPAMBWSM_EL1.MAX.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The value set in the MAX field must be less than or equal to this upper bound.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBWCAP_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBWCAP_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b110

```
if !(IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x910];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMBWCAP_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBWCAP_EL2;
```

```
MSR MPAMBWCAP_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b110

```
if !(IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x910] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMBWCAP_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBWCAP_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMBWIDR_EL1, MPAM PE-side Bandwidth Controls ID Register

The MPAMBWIDR_EL1 characteristics are:

Purpose

Indicates the supported PE-side memory bandwidth parameter values.

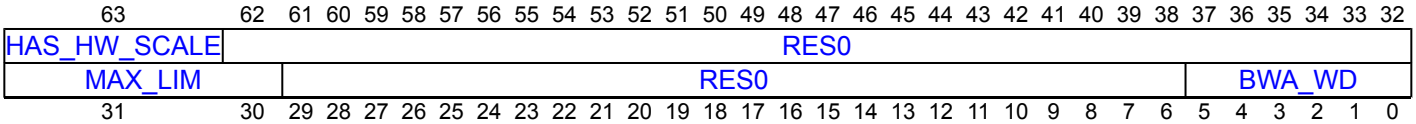
Configuration

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented. Otherwise, direct accesses to MPAMBWIDR_EL1 are UNDEFINED.

Attributes

MPAMBWIDR_EL1 is a 64-bit register.

Field descriptions



HAS_HW_SCALE, bit [63]

Indicates whether hardware support for auto-scaling of MPAMBWn_ELx.MAX, MPAMBWSM_EL1.MAX and MPAMBWCAP_EL2.CAP limits is available.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_HW_SCALE	Meaning
0b0	Hardware support for auto-scaling is not implemented.
0b1	Hardware support for auto-scaling is implemented.

Access to this field is **RO**.

Bits [62:32]

Reserved, RES0.

MAX_LIM, bits [31:30]

Indicates the implemented maximum-bandwidth limit partitioning behaviors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MAX_LIM	Meaning
0b00	Both soft limit and hard limit behaviors are implemented.
0b01	Soft limit behavior is implemented.
0b10	Hard limit behavior is implemented.
0b11	Reserved.

Access to this field is **RO**.

Bits [29:6]

Reserved, RES0.

BWA_WD, bits [5:0]

Indicates the number of implemented bits in the bandwidth allocation fields MPAMBWn_ELx.MAX, MPAMBWSM_EL1.MAX and MPAMBWCAP_EL2.CAP.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BWA_WD	Meaning
0b000001..0b010000	Number of implemented bits in the bandwidth allocation fields.

Access to this field is **RO**.

Accessing MPAMBWIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBWIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBWIDR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAMBWIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMBWIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBWIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMBWSM_EL1, MPAM Streaming Mode Bandwidth Control Register (EL1)

The MPAMBWSM_EL1 characteristics are:

Purpose

Enables software to configure a maximum fraction of memory bandwidth that the PE is permitted for SME memory accesses labelled with values from MPAMSM_EL1.

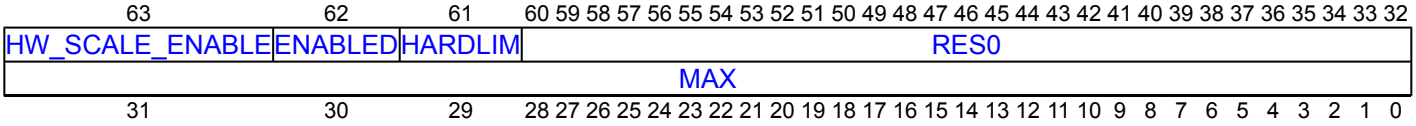
Configuration

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented and FEAT_SME is implemented. Otherwise, direct accesses to MPAMBWSM_EL1 are UNDEFINED.

Attributes

MPAMBWSM_EL1 is a 64-bit register.

Field descriptions



HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == 1:

Enables hardware bandwidth scaling of the MPAMBWSM_EL1.MAX value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling for streaming PARTIDs is disabled.
0b1	PE-side memory bandwidth control hardware scaling for streaming PARTIDs is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control for streaming PARTIDs.

ENABLED	Meaning
0b0	PE-side memory bandwidth control for streaming PARTIDs is disabled.
0b1	PE-side memory bandwidth control for streaming PARTIDs is enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

HARDLIM, bit [61]

PE-side Maximum-bandwidth Limit Behavior Selection.

HARDLIM	Meaning
0b0	Soft limit: when MPAMBWSM_EL1.MAX bandwidth is exceeded, the PE is unregulated unless the downstream memory path is saturated. It is IMPLEMENTATION DEFINED how hardware determines when the downstream memory path is saturated.
0b1	Hard limit: when MPAMBWSM_EL1.MAX bandwidth is exceeded, the PE does not use any more bandwidth until the memory bandwidth for the PE falls below MPAMBWSM_EL1.MAX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

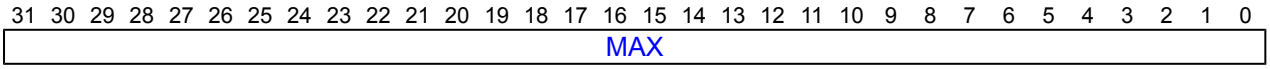
- When MPAMBWIDR_EL1.MAX_LIM == 0b00, access to this field is **RW**.
- When MPAMBWIDR_EL1.MAX_LIM == 0b01, access to this field is **RAZ/WI**.
- When MPAMBWIDR_EL1.MAX_LIM == 0b10, access to this field is **RAO/WI**.

Bits [60:32]

Reserved, RES0.

MAX, bits [31:0]

MAX encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 1 and MPAMBWSM_EL1.HW_SCALE_ENABLE == 1



MAX, bits [31:0]

Maximum memory bandwidth allocated to the partition selected by MPAMSM_EL1.PARTID_D.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

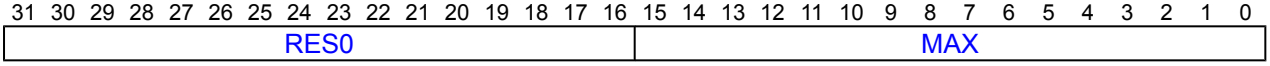
Bits [31:16] represent the integer part of the value.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MAX encoding when MPAMBWIDR_EL1.HAS_HW_SCALE == 0 or MPAMBWSM_EL1.HW_SCALE_ENABLE == 0



Bits [31:16]

Reserved, RES0.

MAX, bits [15:0]

Maximum memory bandwidth allocated to the partition selected by MPAMSM_EL1.PARTID_D.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBWSM_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBWSM_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b111

```
if !(IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) && IsFeatureImplemented(FEAT_SME)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBWSM_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAMBWSM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMBWSM_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBWSM_EL1;
```

```
MSR MPAMBWSM_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b111

```
if !(IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) && IsFeatureImplemented(FEAT_SME)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBWSM_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        MPAMBWSM_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMBWSM_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBWSM_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMHCR_EL2, MPAM Hypervisor Control Register (EL2)

The MPAMHCR_EL2 characteristics are:

Purpose

Controls the PARTID virtualization features of MPAM. It controls the mapping of virtual PARTIDs into physical PARTIDs in MPAM0_EL1 when EL0_VPMEN == 1 and in MPAM1_EL1 when EL1_VPMEN == 1.

Configuration

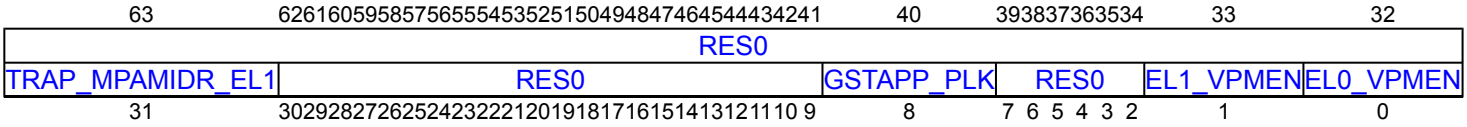
This register is present only when FEAT_MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMHCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMHCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TRAP_MPAMIDR_EL1, bit [31]

Trap accesses from EL1 to MPAMIDR_EL1 to EL2.

TRAP_MPAMIDR_EL1	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Direct accesses to MPAMIDR_EL1 from EL1 are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When EL3 is not implemented, this field resets to '1'.
  - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

Bits [30:9]

Reserved, RES0.

GSTAPP_PLK, bit [8]

Make the PARTIDs at EL0 the same as the PARTIDs at EL1. When executing at EL0, EL2 is enabled, HCR_EL2.TGE == 0 and GSTAPP_PLK = 1, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM labels for memory requests.

GSTAPP_PLK	Meaning
0b0	MPAM0_EL1 is used to generate MPAM labels when executing at EL0.
0b1	MPAM1_EL1 is used to generate MPAM labels when executing at EL0 with EL2 enabled and HCR_EL2.TGE == 0. Otherwise MPAM0_EL1 is used.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:2]

Reserved, RES0.

EL1_VPMEN, bit [1]

Enable the virtual PARTID mapping of the PARTID fields in MPAM1_EL1 when executing at EL1. This bit also enables virtual PARTID mapping when MPAM1_EL1 is used to generate MPAM labels for memory requests at EL0 due to GSTAPP_PLK == 1.

EL1_VPMEN	Meaning
0b0	MPAM1_EL1.PARTID_I and MPAM1_EL1.PARTID_D are physical PARTIDs that are used to label memory system requests.
0b1	MPAM1_EL1.PARTID_I and MPAM1_EL1.PARTID_D are virtual PARTIDs that are used to index the PhyPARTID fields of MPAMVPM0_EL2 to MPAMVPM7_EL2 registers to map the virtual PARTID into a physical PARTID to label memory system requests.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0_VPMEN, bit [0]

Enable the virtual PARTID mapping of the PARTID fields of MPAM0_EL1 unless the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL0_VPMEN is ignored and MPAM0_EL1 PARTID fields are not mapped.

When MPAMHCR_EL2.GSTAPP_PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used as the source of PARTIDs and the virtual PARTID mapping of MPAM1_EL1 PARTIDs is controlled by MPAMHCR_EL2.EL1_VPMEN.

EL0_VPMEN	Meaning
0b0	MPAM0_EL1.PARTID_I and MPAM0_EL1.PARTID_D are physical PARTIDs that are used to label memory system requests.
0b1	MPAM0_EL1.PARTID_I and MPAM0_EL1.PARTID_D are virtual PARTIDs that are used to index the PhyPARTID fields of MPAMVPM0_EL2 to MPAMVPM7_EL2 registers to map the virtual PARTID into a physical PARTID to label memory system requests.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMHCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMHCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x930];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMHCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMHCR_EL2;
```

MSR MPAMHCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x930] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMHCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMHCR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMIDR_EL1, MPAM ID Register (EL1)

The MPAMIDR_EL1 characteristics are:

Purpose

Indicates the presence and maximum PARTID and PMG values supported in the implementation. It also indicates whether the implementation supports MPAM virtualization.

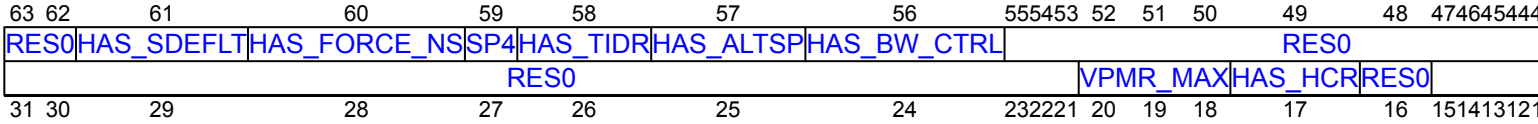
Configuration

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMIDR_EL1 are UNDEFINED.

Attributes

MPAMIDR_EL1 is a 64-bit register.

Field descriptions



MPAMIDR_EL1 indicates the MPAM implementation parameters of the PE.

Bits [63:62]

Reserved, RES0.

HAS_SDEFLT, bit [61]

HAS_SDEFLT indicates support for MPAM3_EL3.SDEFLT bit.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_SDEFLT	Meaning
0b0	The SDEFLT bit is not implemented in MPAM3_EL3.
0b1	The SDEFLT bit is implemented in MPAM3_EL3.

When MPAM3_EL3.SDEFLT == 1, accesses from the Secure Execution state use the default PARTID, PARTID == 0.

Access to this field is **RO**.

HAS_FORCE_NS, bit [60]

HAS_FORCE_NS indicates support for MPAM3_EL3.FORCE_NS bit.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_FORCE_NS	Meaning
0b0	The FORCE_NS bit is not implemented in MPAM3_EL3.
0b1	The FORCE_NS bit is implemented in MPAM3_EL3.

When MPAM3_EL3.FORCE_NS == 1, accesses from the Secure Execution state have MPAM_NS == 1.

Access to this field is **RO**.

SP4, bit [59]

Supports 4 MPAM PARTID spaces.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SP4	Meaning
0b0	MPAM supports 2 PARTID spaces.
0b1	MPAM supports 4 PARTID spaces.

Access to this field is **RO**.

HAS_TIDR, bit [58]

HAS_TIDR indicates support for MPAM2_EL2.TIDR bit.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_TIDR	Meaning
0b0	The TIDR bit is not implemented in MPAM2_EL2.
0b1	The TIDR bit is implemented in MPAM2_EL2.

Note

Arm recommends that when the MPAM version is MPAM v0.1 or MPAM v1.1, MPAMIDR_EL1.HAS_TIDR is 1 and that the MPAM2_EL2.TIDR field is implemented.

Access to this field is **RO**.

HAS_ALTSP, bit [57]

HAS_ALTSP indicates support for alternative PARTID spaces.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_ALTSP	Meaning
0b0	Alternative PARTID spaces are not implemented.
0b1	Alternative PARTID spaces are implemented with control bits in MPAM3_EL3 and MPAM2_EL2.

Access to this field is **RO**.

HAS_BW_CTRL, bit [56]

HAS_BW_CTRL indicates support for PE-side bandwidth controls.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_BW_CTRL	Meaning
0b0	PE-side bandwidth controls are not implemented.
0b1	PE-side bandwidth controls are implemented.

FEAT_MPAM_PE_BW_CTRL implements the functionality identified by the value 0b1.

Access to this field is **RO**.

Bits [55:40]

Reserved, RES0.

PMG_MAX, bits [39:32]

The largest value of PMG that the implementation can generate. The PMG_I and PMG_D fields of every MPAMn_ELx must implement at least enough bits to represent PMG_MAX.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [31:21]

Reserved, RES0.

VPMR_MAX, bits [20:18]

When MPAMIDR_EL1.HAS_HCR == 1:

Indicates the maximum register index n for the MPAMVPM<n>_EL2 registers.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

HAS_HCR, bit [17]

HAS_HCR indicates that the PE implementation supports MPAM virtualization, including MPAMHCR_EL2, MPAMVPMV_EL2, and MPAMVPM<n>_EL2 with n in the range 0 to VPMR_MAX. Must be 0 if EL2 is not implemented in either Security state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_HCR	Meaning
0b0	MPAM virtualization is not supported.
0b1	MPAM virtualization is supported.

Access to this field is **RO**.

Bit [16]

Reserved, RES0.

PARTID_MAX, bits [15:0]

The largest value of PARTID that the implementation can generate. The PARTID_I and PARTID_D fields of every MPAMn_ELx must implement at least enough bits to represent PARTID_MAX.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPAMIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMIDR_EL1
```

op0	op1	CRn	CRm	op2

0b11	0b000	0b1010	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMIDR_EL1.HAS_HCR == '1' && MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MPAMIDR_EL1.HAS_TIDR == '1' && MPAM2_EL2.TIDR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAMIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMSM_EL1, MPAM Streaming Mode Register

The MPAMSM_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests that are:

- Issued due to the execution of SME load and store instructions.
- Issued when the PE is in Streaming SVE mode due to the execution of SVE and SIMD&FP load and store instructions and SVE prefetch instructions.

If an implementation uses a shared SMCU, then the MPAM labels in this register have precedence over the labels in MPAM0_EL1, MPAM1_EL1, MPAM2_EL2, and MPAM3_EL3.

If an implementation includes an SMCU that is not shared with other PEs, then it is IMPLEMENTATION DEFINED whether the MPAM labels in this register have precedence over the labels in MPAM0_EL1, MPAM1_EL1, MPAM2_EL2, and MPAM3_EL3.

The MPAM labels in this register are only used if MPAM1_EL1.MPAMEN is 1.

For memory requests issued from EL0, the MPAM PARTID in this register is virtual and mapped into a physical PARTID when all of the following are true:

- EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The MPAM virtualization option is implemented and MPAMHCR_EL2.EL0_VPMEN is 1.

For memory requests issued from EL1, the MPAM PARTID in this register is virtual and mapped into a physical PARTID when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The MPAM virtualization option is implemented and MPAMHCR_EL2.EL1_VPMEN is 1.

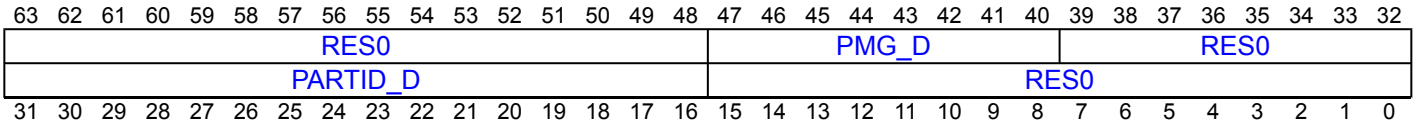
Configuration

This register is present only when FEAT_MPAM is implemented and FEAT_SME is implemented. Otherwise, direct accesses to MPAMSM_EL1 are UNDEFINED.

Attributes

MPAMSM_EL1 is a 64-bit register.

Field descriptions



Bits [63:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [39:32]

Reserved, RES0.

PARTID_D, bits [31:16]

Partition ID for requests issued due to the execution at any Exception level of SME load and store instructions and, when the PE is in Streaming SVE mode, SVE and SIMD&FP load and store instructions and SVE prefetch instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:0]

Reserved, RES0.

Accessing MPAMSM_EL1

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMSM_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b011

```
if !(IsFeatureImplemented(FEAT_MPAM) && IsFeatureImplemented(FEAT_SME)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.EnMPAMSM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAMSM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMSM_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMSM_EL1;
```

```
MSR MPAMSM_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b011

```
if !(IsFeatureImplemented(FEAT_MPAM) && IsFeatureImplemented(FEAT_SME)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.EnMPAMSM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        MPAMSM_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMSM_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMSM_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM0_EL2, MPAM Virtual PARTID Mapping Register 0

The MPAMVPM0_EL2 characteristics are:

Purpose

MPAMVPM0_EL2 provides mappings from virtual PARTIDs 0 - 3 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 register. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

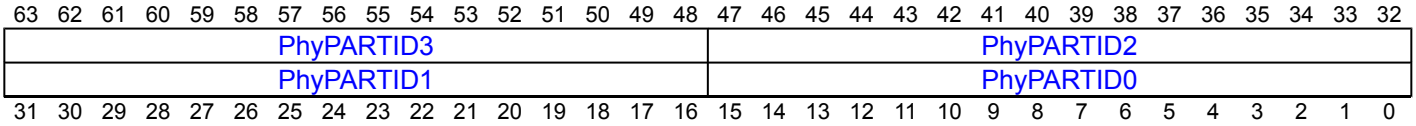
This register is present only when FEAT_MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMVPM0_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM0_EL2 is a 64-bit register.

Field descriptions



PhyPARTID3, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 3. PhyPARTID3 gives the mapping of virtual PARTID 3 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID2, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 2. PhyPARTID2 gives the mapping of virtual PARTID 2 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID1, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 1. PhyPARTID1 gives the mapping of virtual PARTID 1 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID0, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 0. PhyPARTID0 gives the mapping of virtual PARTID 0 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM0_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x940];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM0_EL2;
```

```
MSR MPAMVPM0_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x940] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM0_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM0_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM1_EL2, MPAM Virtual PARTID Mapping Register 1

The MPAMVPM1_EL2 characteristics are:

Purpose

MPAMVPM1_EL2 provides mappings from virtual PARTIDs 4 - 7 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM0_EL2 to MPAMVPM7_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

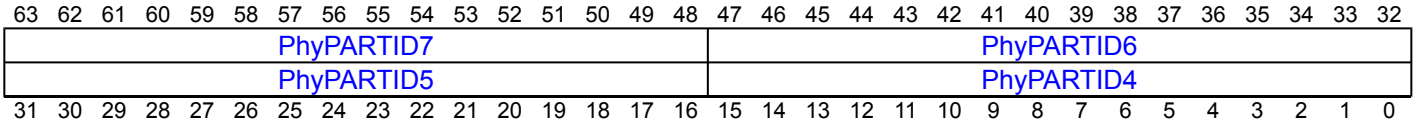
This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1, and UInt(MPAMIDR_EL1.VPMR_MAX) > 0. Otherwise, direct accesses to MPAMVPM1_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM1_EL2 is a 64-bit register.

Field descriptions



PhyPARTID7, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 7. PhyPARTID7 gives the mapping of virtual PARTID 7 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID6, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 6. PhyPARTID6 gives the mapping of virtual PARTID 6 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID5, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 5. PhyPARTID5 gives the mapping of virtual PARTID 5 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID4, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 4. PhyPARTID4 gives the mapping of virtual PARTID 4 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM1_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM1_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x948];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM1_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM1_EL2;
```

```
MSR MPAMVPM1_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x948] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM1_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM1_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM2_EL2, MPAM Virtual PARTID Mapping Register 2

The MPAMVPM2_EL2 characteristics are:

Purpose

MPAMVPM2_EL2 provides mappings from virtual PARTIDs 8 - 11 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM0_EL2 to MPAMVPM7_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

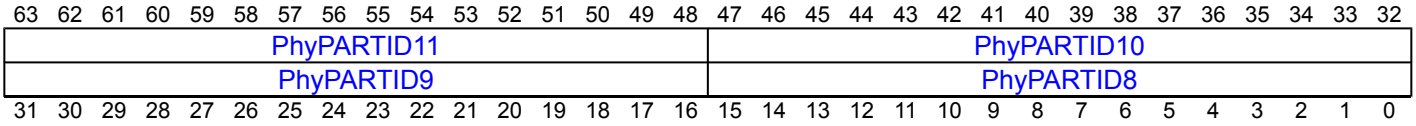
This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1, and UInt(MPAMIDR_EL1.VPMR_MAX) > 1. Otherwise, direct accesses to MPAMVPM2_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM2_EL2 is a 64-bit register.

Field descriptions



PhyPARTID11, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 11. PhyPARTID11 gives the mapping of virtual PARTID 11 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID10, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 10. PhyPARTID10 gives the mapping of virtual PARTID 10 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID9, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 9. PhyPARTID9 gives the mapping of virtual PARTID 9 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID8, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 8. PhyPARTID8 gives the mapping of virtual PARTID 8 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM2_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b010

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x950];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM2_EL2;
```

```
MSR MPAMVPM2_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b010

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x950] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM2_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM3_EL2, MPAM Virtual PARTID Mapping Register 3

The MPAMVPM3_EL2 characteristics are:

Purpose

MPAMVPM3_EL2 provides mappings from virtual PARTIDs 12 - 15 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

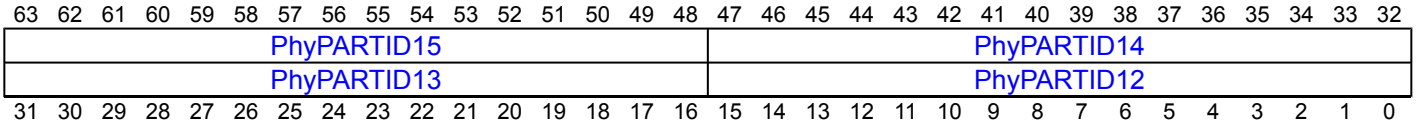
This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1, and UInt(MPAMIDR_EL1.VPMR_MAX) > 2. Otherwise, direct accesses to MPAMVPM3_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM3_EL2 is a 64-bit register.

Field descriptions



PhyPARTID15, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 15. PhyPARTID15 gives the mapping of virtual PARTID 15 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID14, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 14. PhyPARTID14 gives the mapping of virtual PARTID 14 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID13, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 13. PhyPARTID13 gives the mapping of virtual PARTID 13 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID12, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 12. PhyPARTID12 gives the mapping of virtual PARTID 12 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM3_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM3_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b011

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x958];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM3_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM3_EL2;
```

```
MSR MPAMVPM3_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b011

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x958] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM3_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM3_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM4_EL2, MPAM Virtual PARTID Mapping Register 4

The MPAMVPM4_EL2 characteristics are:

Purpose

MPAMVPM4_EL2 provides mappings from virtual PARTIDs 16 - 19 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

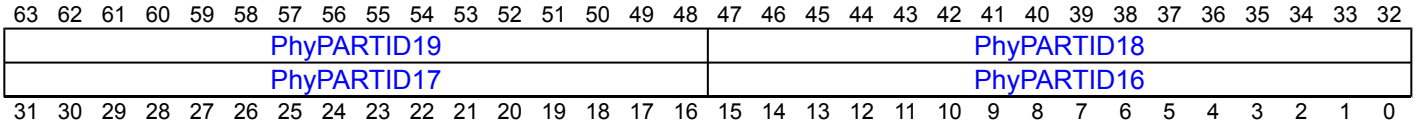
This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1, and UInt(MPAMIDR_EL1.VPMR_MAX) > 3. Otherwise, direct accesses to MPAMVPM4_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM4_EL2 is a 64-bit register.

Field descriptions



PhyPARTID19, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 19. PhyPARTID19 gives the mapping of virtual PARTID 19 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID18, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 18. PhyPARTID18 gives the mapping of virtual PARTID 18 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID17, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 17. PhyPARTID17 gives the mapping of virtual PARTID 17 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID16, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 16. PhyPARTID16 gives the mapping of virtual PARTID 16 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM4_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM4_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b100

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
3) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x960];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM4_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM4_EL2;
```

```
MSR MPAMVPM4_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b100

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
3) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x960] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM4_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM4_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM5_EL2, MPAM Virtual PARTID Mapping Register 5

The MPAMVPM5_EL2 characteristics are:

Purpose

MPAMVPM5_EL2 provides mappings from virtual PARTIDs 20 - 23 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

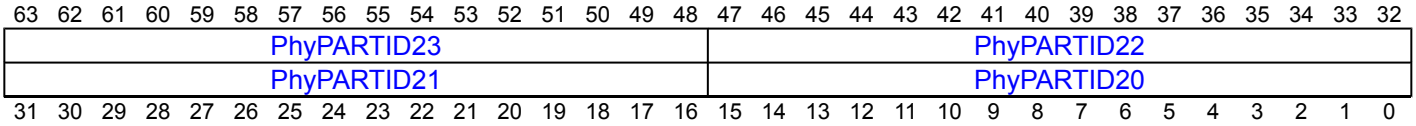
This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1, and UInt(MPAMIDR_EL1.VPMR_MAX) > 4. Otherwise, direct accesses to MPAMVPM5_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM5_EL2 is a 64-bit register.

Field descriptions



PhyPARTID23, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 23. PhyPARTID23 gives the mapping of virtual PARTID 23 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID22, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 22. PhyPARTID22 gives the mapping of virtual PARTID 22 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID21, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 21. PhyPARTID21 gives the mapping of virtual PARTID 21 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID20, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 20. PhyPARTID20 gives the mapping of virtual PARTID 20 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM5_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM5_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
4) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x968];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM5_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM5_EL2;
```

```
MSR MPAMVPM5_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
4) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x968] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM5_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM5_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM6_EL2, MPAM Virtual PARTID Mapping Register 6

The MPAMVPM6_EL2 characteristics are:

Purpose

MPAMVPM6_EL2 provides mappings from virtual PARTIDs 24 - 27 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

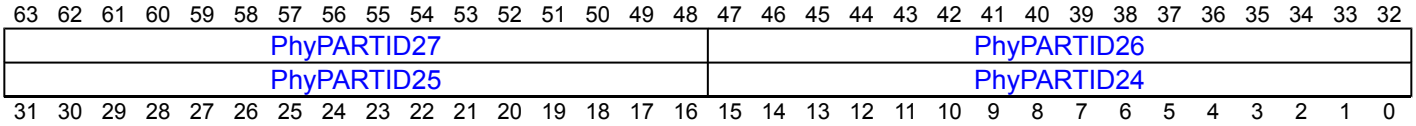
This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1, and UInt(MPAMIDR_EL1.VPMR_MAX) > 5. Otherwise, direct accesses to MPAMVPM6_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM6_EL2 is a 64-bit register.

Field descriptions



PhyPARTID27, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 27. PhyPARTID27 gives the mapping of virtual PARTID 27 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID26, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 26. PhyPARTID26 gives the mapping of virtual PARTID 26 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID25, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 25. PhyPARTID25 gives the mapping of virtual PARTID 25 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID24, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 24. PhyPARTID24 gives the mapping of virtual PARTID 24 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM6_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM6_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b110

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
5) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x970];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM6_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM6_EL2;
```

```
MSR MPAMVPM6_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b110

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX) >
5) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x970] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM6_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM6_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM7_EL2, MPAM Virtual PARTID Mapping Register 7

The MPAMVPM7_EL2 characteristics are:

Purpose

MPAMVPM7_EL2 provides mappings from virtual PARTIDs 28 - 31 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

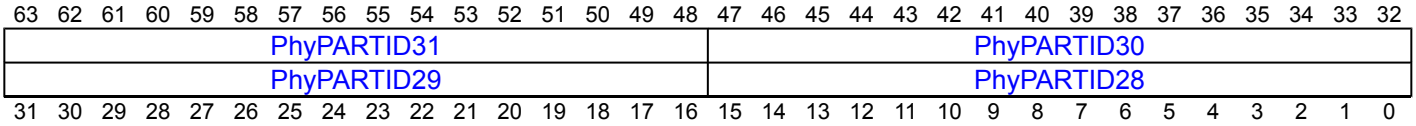
This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1, and UInt(MPAMIDR_EL1.VPMR_MAX) == 7. Otherwise, direct accesses to MPAMVPM7_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM7_EL2 is a 64-bit register.

Field descriptions



PhyPARTID31, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 31. PhyPARTID31 gives the mapping of virtual PARTID 31 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID30, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 30. PhyPARTID30 gives the mapping of virtual PARTID 30 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID29, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 29. PhyPARTID29 gives the mapping of virtual PARTID 29 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID28, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 28. PhyPARTID28 gives the mapping of virtual PARTID 28 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM7_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM7_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b111

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX)
== 7) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x978];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM7_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM7_EL2;
```

```
MSR MPAMVPM7_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b111

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' && UInt(MPAMIDR_EL1.VPMR_MAX)
== 7) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x978] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM7_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM7_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPMV_EL2, MPAM Virtual Partition Mapping Valid Register

The MPAMVPMV_EL2 characteristics are:

Purpose

Valid bits for virtual PARTID mapping entries. Each bit m corresponds to virtual PARTID mapping entry m in the MPAMVPM<n>_EL2 registers where n = m >> 2.

Configuration

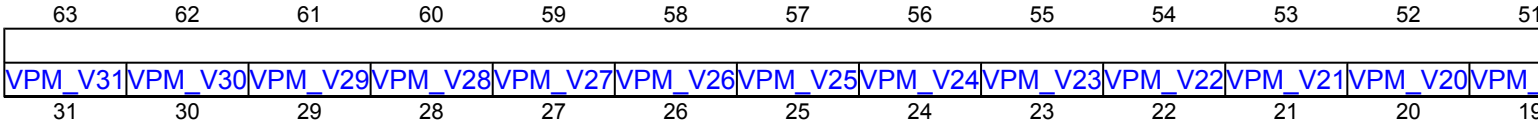
This register is present only when FEAT_MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMVPMV_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPMV_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

VPM_V<m>, bit [m], for m = 31 to 0

Contains valid bit for virtual PARTID mapping entry corresponding to virtual PARTID<m>.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPMV_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPMV_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x938];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPMV_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPMV_EL2;
```

MSR MPAMVPMV_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x938] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPMV_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPMV_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPIDR_EL1, Multiprocessor Affinity Register

The MPIDR_EL1 characteristics are:

Purpose

In a multiprocessor system, provides an additional PE identification mechanism.

Configuration

AArch64 System register MPIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MPIDR[31:0].

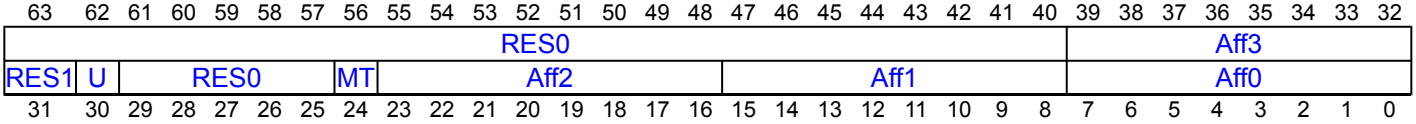
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MPIDR_EL1 are UNDEFINED.

In a uniprocessor system, Arm recommends that each Aff<n> field of this register returns a value of 0.

Attributes

MPIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3. See the description of Aff0 for more information.

Aff3 is not supported in AArch32 state.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [31]

Reserved, RES1.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Access to this field is **RO**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is **RO**.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.MPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() then
        X[t, 64] = VMPIDR_EL2;
    else
        X[t, 64] = MPIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MPIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR0_EL1, AArch32 Media and VFP Feature Register 0

The MVFR0_EL1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Must be interpreted with MVFR1_EL1 and MVFR2_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register MVFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register MVFR0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MVFR0_EL1 are UNDEFINED.

In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

Attributes

MVFR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
FPRound				FPShVec				FPSqrt				FPDivide				FPTrap				FPDP				FPSP				SIMDReg			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:32]

Reserved, RES0.

FPRound, bits [31:28]

Floating-Point Rounding modes. Indicates whether the floating-point implementation provides support for rounding modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPRound	Meaning
0b0000	Not implemented, or only Round to Nearest mode supported, except that Round towards Zero mode is supported for VCVT instructions that always use that rounding mode regardless of the FPSCR setting.
0b0001	All rounding modes supported.

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

FPShVec, bits [27:24]

Short Vectors. Indicates whether the floating-point implementation provides support for the use of short vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPShVec	Meaning
0b0000	Short vectors not supported.
0b0001	Short vector operation supported.

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

Access to this field is **RO**.

FPSqrt, bits [23:20]

Square Root. Indicates whether the floating-point implementation provides support for the ARMv6 VFP square root operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPSqrt	Meaning
0b0000	Not supported in hardware.
0b0001	Supported.

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

The VSQRT.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VSQRT.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

Access to this field is **RO**.

FPDivide, bits [19:16]

Indicates whether the floating-point implementation provides support for VFP divide operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPDivide	Meaning
0b0000	Not supported in hardware.
0b0001	Supported.

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

The VDIV.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VDIV.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

Access to this field is **RO**.

FPTrap, bits [15:12]

Floating-Point Exception Trapping. Indicates whether the floating-point implementation provides support for exception trapping.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPTrap	Meaning
0b0000	Not supported.
0b0001	Supported.

All other values are reserved.

A value of 0b0001 indicates that, when the corresponding trap is enabled, a floating-point exception generates an exception.

Access to this field is **RO**.

FPDP, bits [11:8]

Floating-Point Double-Precision. Indicates whether the floating-point implementation provides support for double-precision operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPDP	Meaning
0b0000	Not supported in hardware.
0b0001	Supported, VFPv2.
0b0010	Supported, VFPv3, VFPv4, or Armv8. VFPv3 and Armv8 add an instruction to load a double-precision floating-point constant, and conversions between double-precision and fixed-point values.

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP double-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F64 is only available if the Square root field is 0b0001.
- VDIV.F64 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the single-precision field is nonzero.

Access to this field is **RO**.

FPSP, bits [7:4]

Floating-Point Single-Precision. Indicates whether the floating-point implementation provides support for single-precision operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPSP	Meaning
0b0000	Not supported in hardware.
0b0001	Supported, VFPv2.
0b0010	Supported, VFPv3 or VFPv4. VFPv3 adds an instruction to load a single-precision floating-point constant, and conversions between single-precision and fixed-point values.

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP single-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F32 is only available if the Square root field is 0b0001.
- VDIV.F32 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the double-precision field is nonzero.

Access to this field is **RO**.

SIMDReg, bits [3:0]

Advanced SIMD registers. Indicates whether the Advanced SIMD and floating-point implementation provides support for the Advanced SIMD and floating-point register bank.

The value of this field is an IMPLEMENTATION DEFINED choice of:

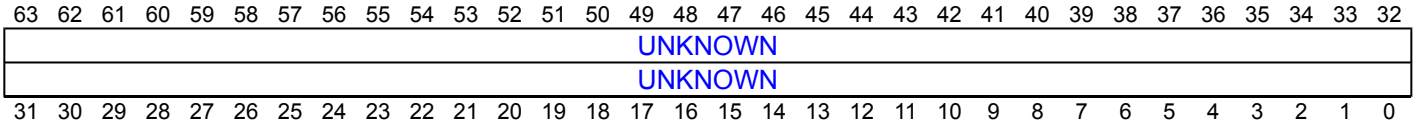
SIMDReg	Meaning
0b0000	The implementation has no Advanced SIMD and floating-point support.
0b0001	The implementation includes floating-point support with 16 x 64-bit registers.
0b0010	The implementation includes Advanced SIMD and floating-point support with 32 x 64-bit registers.

All other values are reserved.

In Armv8-A the permitted values are `0b0000` and `0b0010`.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing MVFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

`MRS <Xt>, MVFR0_EL1`

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MVFR0_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR1_EL1, AArch32 Media and VFP Feature Register 1

The MVFR1_EL1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Must be interpreted with MVFR0_EL1 and MVFR2_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register MVFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register MVFR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MVFR1_EL1 are UNDEFINED.

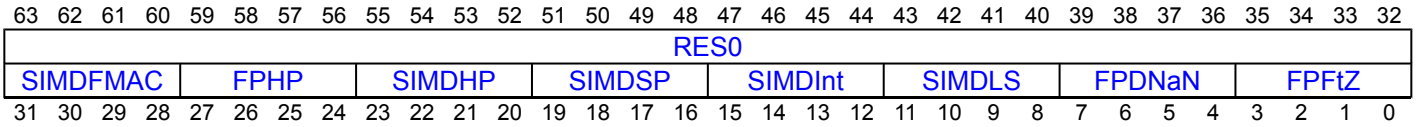
In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

Attributes

MVFR1_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:32]

Reserved, RES0.

SIMDFMAC, bits [31:28]

Advanced SIMD Fused Multiply-Accumulate. Indicates whether the Advanced SIMD implementation provides fused multiply accumulate instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDFMAC	Meaning
0b0000	Not implemented.
0b0001	Implemented.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

The Advanced SIMD and floating-point implementations must provide the same level of support for these instructions.

Access to this field is **RO**.

FPHP, bits [27:24]

Floating-Point Half-Precision. Indicates the level of half-precision floating-point support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPHP	Meaning
0b0000	Not supported.
0b0001	Floating-point half-precision conversion instructions are supported for conversion between single-precision and half-precision.
0b0010	As for 0b0001, and adds instructions for conversion between double-precision and half-precision.
0b0011	As for 0b0010, and adds support for half-precision floating-point arithmetic.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 in an implementation without floating-point support.
- 0b0010 in an implementation with floating-point support that does not include the FEAT_FP16 extension.
- 0b0011 in an implementation with floating-point support that includes the FEAT_FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the SIMDHP field, meaning the permitted values are:

Half-Precision instructions supported	FPHP	SIMDHP
No support	0b0000	0b0000
Conversions only	0b0010	0b0001
Conversions and arithmetic	0b0011	0b0010

Access to this field is **RO**.

SIMDHP, bits [23:20]

Advanced SIMD Half-Precision. Indicates the level of half-precision floating-point support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDHP	Meaning
0b0000	Not supported.
0b0001	SIMD half-precision conversion instructions are supported for conversion between single-precision and half-precision.
0b0010	As for 0b0001, and adds support for half-precision floating-point arithmetic.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 in an implementation without SIMD floating-point support.
- 0b0001 in an implementation with SIMD floating-point support that does not include the FEAT_FP16 extension.
- 0b0010 in an implementation with SIMD floating-point support that includes the FEAT_FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the FPHP field, meaning the permitted values are:

Half-Precision instructions supported	FPHP	SIMDHP
No support	0b0000	0b0000
Conversions only	0b0010	0b0001
Conversions and arithmetic	0b0011	0b0010

Access to this field is **RO**.

SIMDSP, bits [19:16]

Advanced SIMD Single-Precision. Indicates whether the Advanced SIMD and floating-point implementation provides single-precision floating-point instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDSP	Meaning
0b0000	Not implemented.
0b0001	Implemented. This value is permitted only if the SIMDInt field is 0b0001.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

SIMDInt, bits [15:12]

Advanced SIMD Integer. Indicates whether the Advanced SIMD and floating-point implementation provides integer instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDInt	Meaning
0b0000	Not implemented.
0b0001	Implemented.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

SIMDLS, bits [11:8]

Advanced SIMD Load/Store. Indicates whether the Advanced SIMD and floating-point implementation provides load/store instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDLS	Meaning
0b0000	Not implemented.
0b0001	Implemented.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

FPDNaN, bits [7:4]

Default NaN mode. Indicates whether the floating-point implementation provides support only for the Default NaN mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPDNaN	Meaning
0b0000	Not implemented, or hardware supports only the Default NaN mode.
0b0001	Hardware supports propagation of NaN values.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

FPFtZ, bits [3:0]

Flush to Zero mode. Indicates whether the floating-point implementation provides support only for the Flush-to-Zero mode of operation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPFtZ	Meaning
0b0000	Not implemented, or hardware supports only the Flush-to-Zero mode of operation.
0b0001	Hardware supports full denormalized number arithmetic.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

Otherwise:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
UNKNOWN																															
UNKNOWN																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Reserved, UNKNOWN.

Accessing MVFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MVFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MVFR1_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR2_EL1, AArch32 Media and VFP Feature Register 2

The MVFR2_EL1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Must be interpreted with MVFR0_EL1 and MVFR1_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register MVFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register MVFR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MVFR2_EL1 are UNDEFINED.

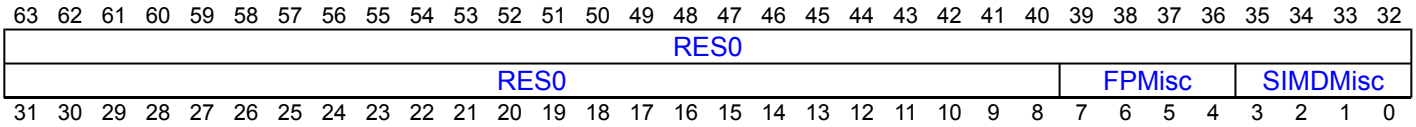
In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

Attributes

MVFR2_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:



Bits [63:8]

Reserved, RES0.

FPMisc, bits [7:4]

Indicates whether the floating-point implementation provides support for miscellaneous VFP features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPMisc	Meaning
0b0000	Not implemented, or no support for miscellaneous features.
0b0001	Support for Floating-point selection.
0b0010	As 0b0001, and Floating-point Conversion to Integer with Directed Rounding modes.
0b0011	As 0b0010, and Floating-point Round to Integer Floating-point.
0b0100	As 0b0011, and Floating-point MaxNum and MinNum.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0100.

Access to this field is **RO**.

SIMDMisc, bits [3:0]

Indicates whether the Advanced SIMD implementation provides support for miscellaneous Advanced SIMD features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

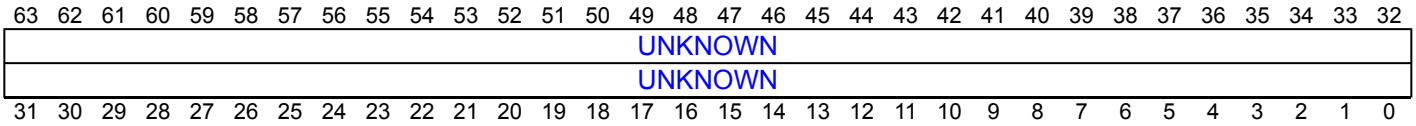
SIMDMisc	Meaning
0b0000	Not implemented, or no support for miscellaneous features.
0b0001	Floating-point Conversion to Integer with Directed Rounding modes.
0b0010	As 0b0001, and Floating-point Round to Integer Floating-point.
0b0011	As 0b0010, and Floating-point MaxNum and MinNum.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0011.

Access to this field is **RO**.

Otherwise:



Bits [63:0]

Reserved, UNKNOWN.

Accessing MVFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MVFR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MVFR2_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

NZCV, Condition Flags

The NZCV characteristics are:

Purpose

Allows access to the condition flags.

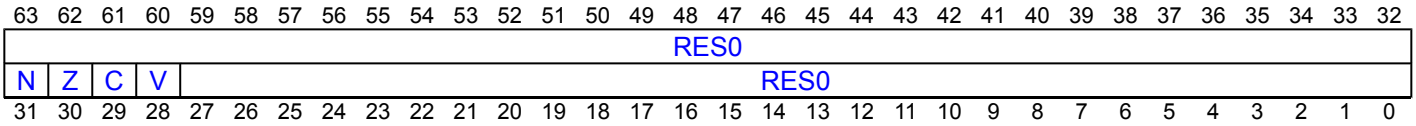
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to NZCV are UNDEFINED.

Attributes

NZCV is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative condition flag. Set to 1 if the result of the last flag-setting instruction was negative.

Z, bit [30]

Zero condition flag. Set to 1 if the result of the last flag-setting instruction was zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

C, bit [29]

Carry condition flag. Set to 1 if the last flag-setting instruction resulted in a carry condition, for example an unsigned overflow on an addition.

V, bit [28]

Overflow condition flag. Set to 1 if the last flag-setting instruction resulted in an overflow condition, for example a signed overflow on an addition.

Bits [27:0]

Reserved, RES0.

Accessing NZCV

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, NZCV

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    X[t, 64] = Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);
elsif PSTATE.EL == EL1 then
    X[t, 64] = Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);
```

MSR NZCV, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    PSTATE.<N,Z,C,V> = X[t, 64]<31:28>;
elsif PSTATE.EL == EL1 then
    PSTATE.<N,Z,C,V> = X[t, 64]<31:28>;
elsif PSTATE.EL == EL2 then
    PSTATE.<N,Z,C,V> = X[t, 64]<31:28>;
elsif PSTATE.EL == EL3 then
    PSTATE.<N,Z,C,V> = X[t, 64]<31:28>;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSDLR_EL1, OS Double Lock Register

The OSDLR_EL1 characteristics are:

Purpose

Used to control the OS Double Lock.

Configuration

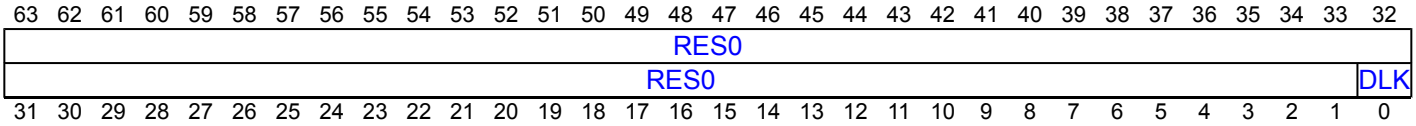
AArch64 System register OSDLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSDLR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSDLR_EL1 are UNDEFINED.

Attributes

OSDLR_EL1 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

DLK, bit [0]

When FEAT_DoubleLock is implemented:

OS Double Lock control bit.

DLK	Meaning
0b0	OS Double Lock unlocked.
0b1	OS Double Lock locked, if DBGPRCR_EL1.CORENPDRQ (Core no powerdown request) bit is set to 0 and the PE is in Non-debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RAZ/WI.

Accessing OSDLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, OSDLR_EL1

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_DoubleLock) && HDFGRTR_EL2.OSDLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' && (IsFeatureImplemented(FEAT_DoubleLock)
|| boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL2.TDOSA") then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = OSDLR_EL1;
```

MSR OSDLR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_DoubleLock) && HDFGWTR_EL2.OSDLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' && (IsFeatureImplemented(FEAT_DoubleLock)
|| boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL2.TDOSA") then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    OSDLR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSDTRRX_EL1, OS Lock Data Transfer Register, Receive

The OSDTRRX_EL1 characteristics are:

Purpose

Used for save and restore of DBGDTRRX_EL0. It is a component of the Debug Communications Channel.

Configuration

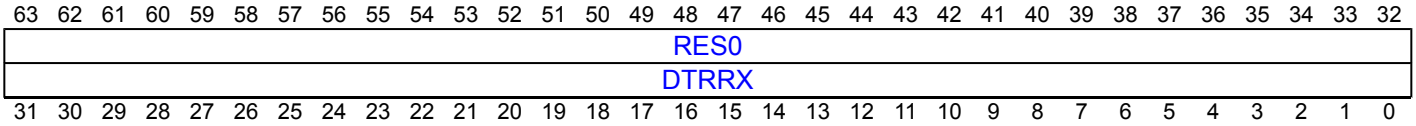
AArch64 System register OSDTRRX_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRRXext[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSDTRRX_EL1 are UNDEFINED.

Attributes

OSDTRRX_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

DTRRX, bits [31:0]

Update DTRRX without side-effect.

Writes to this register update the value in DTRRX and do not change RXfull.

Reads of this register return the last value written to DTRRX and do not change RXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

Accessing OSDTRRX_EL1

Arm deprecates reads and writes of OSDTRRX_EL1 when the OS Lock is unlocked.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, OSDTRRX_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    X[t, 64] = OSDTRRX_EL1;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDTRRX_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDTRRX_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = OSDTRRX_EL1;
```

```
MSR OSDTRRX_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    OSDTRRX_EL1 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRRX_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRRX_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    OSDTRRX_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSDTRTX_EL1, OS Lock Data Transfer Register, Transmit

The OSDTRTX_EL1 characteristics are:

Purpose

Used for save/restore of DBGDTRTX_EL0. It is a component of the Debug Communications Channel.

Configuration

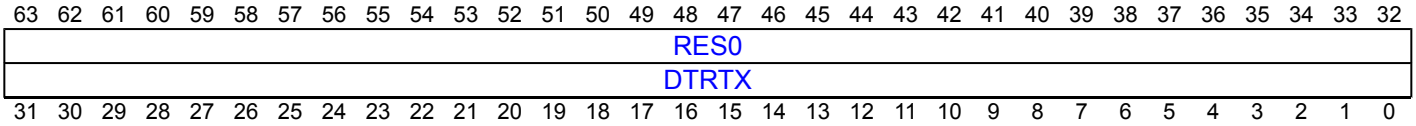
AArch64 System register OSDTRTX_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXext[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSDTRTX_EL1 are UNDEFINED.

Attributes

OSDTRTX_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

DTRTX, bits [31:0]

Return DTRTX without side-effect.

Reads of this register return the value in DTRTX and do not change TXfull.

Writes of this register update the value in DTRTX and do not change TXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

Accessing OSDTRTX_EL1

Arm deprecates reads and writes of OSDTRTX_EL1 when the OS Lock is unlocked.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, OSDTRTX_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    X[t, 64] = OSDTRTX_EL1;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDTRTX_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDTRTX_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = OSDTRTX_EL1;
```

MSR OSDTRTX_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    OSDTRTX_EL1 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRTX_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC ==
'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRTX_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    OSDTRTX_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSECCR_EL1, OS Lock Exception Catch Control Register

The OSECCR_EL1 characteristics are:

Purpose

Provides a mechanism for an operating system to access the contents of EDECCR that are otherwise invisible to software, so it can save/restore the contents of EDECCR over powerdown on behalf of the external debugger.

Configuration

AArch64 System register OSECCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSECCR[31:0].

AArch64 System register OSECCR_EL1 bits [31:0] are architecturally mapped to External register EDECCR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSECCR_EL1 are UNDEFINED.

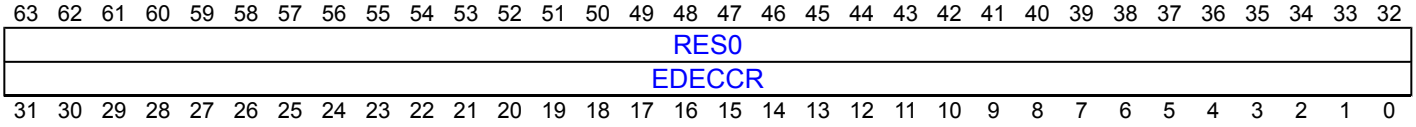
If OSLSR_EL1.OSLK == 0, then OSECCR_EL1 returns an UNKNOWN value on reads and ignores writes.

Attributes

OSECCR_EL1 is a 64-bit register.

Field descriptions

When OSLSR_EL1.OSLK == 1:



Bits [63:32]

Reserved, RES0.

EDECCR, bits [31:0]

Used for save/restore to EDECCR over powerdown.

Reads or writes to this field are indirect accesses to EDECCR.

The reset behavior of this field is:

- On a Cold reset, this field resets to '00000000000000000000000000000000'.

Accessing OSECCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, OSECCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0110	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.OSECCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = OSECCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = OSECCR_EL1;
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = OSECCR_EL1;
```

```
MSR OSECCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0110	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.OSECCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' then
        return;
    else
        OSECCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' then
        return;
    else
        OSECCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' then
        return;
    else
        OSECCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSLAR_EL1, OS Lock Access Register

The OSLAR_EL1 characteristics are:

Purpose

Used to lock or unlock the OS Lock.

Configuration

AArch64 System register OSLAR_EL1 bits [31:0] are architecturally mapped to External register OSLAR_EL1[31:0].

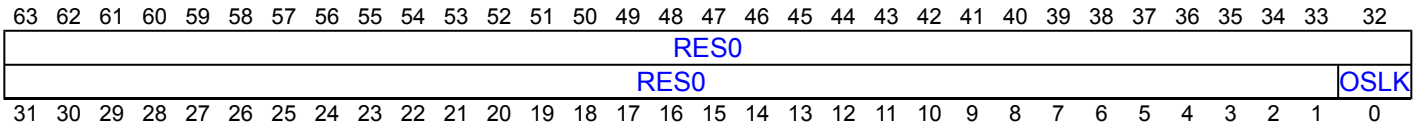
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSLAR_EL1 are UNDEFINED.

The OS Lock can also be locked or unlocked using DBGOSLAR.

Attributes

OSLAR_EL1 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

OSLK, bit [0]

On writes to OSLAR_EL1, bit[0] is copied to the OS Lock.

Use OSLSR_EL1.OSLK to check the current status of the lock.

Accessing OSLAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MSR OSLAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.OSLAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSLAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSLAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    OSLAR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSLSR_EL1, OS Lock Status Register

The OSLSR_EL1 characteristics are:

Purpose

Provides the status of the OS Lock.

Configuration

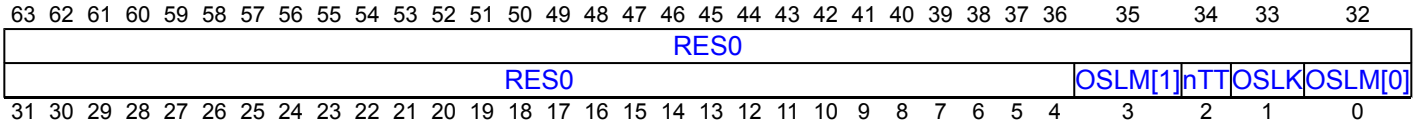
AArch64 System register OSLSR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSLSR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSLSR_EL1 are UNDEFINED.

Attributes

OSLSR_EL1 is a 64-bit register.

Field descriptions



Bits [63:4]

Reserved, RES0.

OSLM, bits [3, 0]

OS Lock model implemented. Identifies the form of OS save and restore mechanism implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OSLM	Meaning
0b00	OS Lock not implemented.
0b10	OS Lock implemented.

All other values are reserved. In an Armv8 implementation the value 0b00 is not permitted.

The OSLM field is split as follows:

- OSLM[1] is OSLSR_EL1[3].
- OSLM[0] is OSLSR_EL1[0].

Access to this field is **RO**.

nTT, bit [2]

Not 32-bit access. This bit is always RAZ. It indicates that a 32-bit access is needed to write the key to the OS Lock Access Register.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

OSLK, bit [1]

OS Lock Status.

OSLK	Meaning
0b0	OS Lock unlocked.
0b1	OS Lock locked.

The OS Lock is locked and unlocked by writing to the OS Lock Access Register.

The reset behavior of this field is:

- On a Cold reset, this field resets to '1'.

Accessing OSLSR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, OSLSR_EL1

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0001	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.OSLSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSLSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSLSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = OSLSR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PAN, Privileged Access Never

The PAN characteristics are:

Purpose

Allows access to the Privileged Access Never bit.

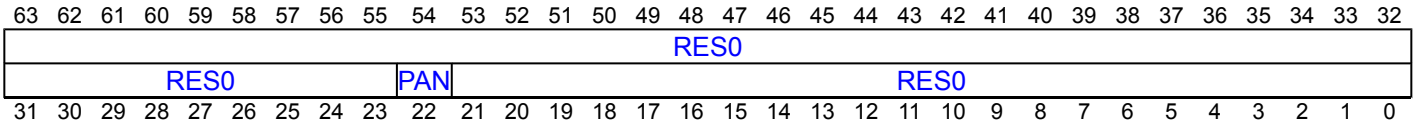
Configuration

This register is present only when FEAT_PAN is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PAN are UNDEFINED.

Attributes

PAN is a 64-bit register.

Field descriptions



Bits [63:23]

Reserved, RES0.

PAN, bit [22]

Privileged Access Never.

PAN	Meaning
0b0	Privileged reads and write are not disabled by this mechanism.
0b1	Disables privileged read and write accesses to addresses accessible at EL0 for an enabled stage 1 translation regime that defines the EL0 permissions.

The value of this bit is usually preserved on taking an exception, except in the following situations:

- When the target of the exception is EL1, and the value of the SCTLR_EL1.SPAN bit is 0, this bit is set to 1.
- When the target of the exception is EL2, the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, and the value of the SCTLR_EL2.SPAN bit is 0, this bit is set to 1.

Bits [21:0]

Reserved, RES0.

Accessing PAN

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PAN
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_PAN) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    X[t, 64] = Zeros(41):PSTATE.PAN:Zeros(22);
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(41):PSTATE.PAN:Zeros(22);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(41):PSTATE.PAN:Zeros(22);
```

MSR PAN, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_PAN) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    PSTATE.PAN = X[t, 64]<22>;
elsif PSTATE.EL == EL2 then
    PSTATE.PAN = X[t, 64]<22>;
elsif PSTATE.EL == EL3 then
    PSTATE.PAN = X[t, 64]<22>;
```

MSR PAN, #<imm>

op0	op1	CRn	op2
0b00	0b000	0b0100	0b100

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PAR_EL1, Physical Address Register

The PAR_EL1 characteristics are:

Purpose

Returns the output address (OA) from an Address translation instruction that executed successfully, or fault information if the instruction did not execute successfully.

Configuration

AArch64 System register PAR_EL1 bits [63:0] are architecturally mapped to AArch32 System register PAR[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to PAR_EL1 are UNDEFINED.

AArch64 System register PAR_EL1 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

Single stage AT Instructions (ATS1*) report their result using the 128-bit format of PAR_EL1 if the translation system that they target uses VMSAv9-128.

ATS12* Instructions report their result using the 128-bit format PAR_EL1 if either of the following is true:

- if stage 2 translations are enabled and the stage 2 translation system uses VMSAv9-128.
- if stage 2 translations are disabled and the stage 1 translation system uses VMSAv9-128.

Otherwise, 64-bit format of PAR_EL1 is used.

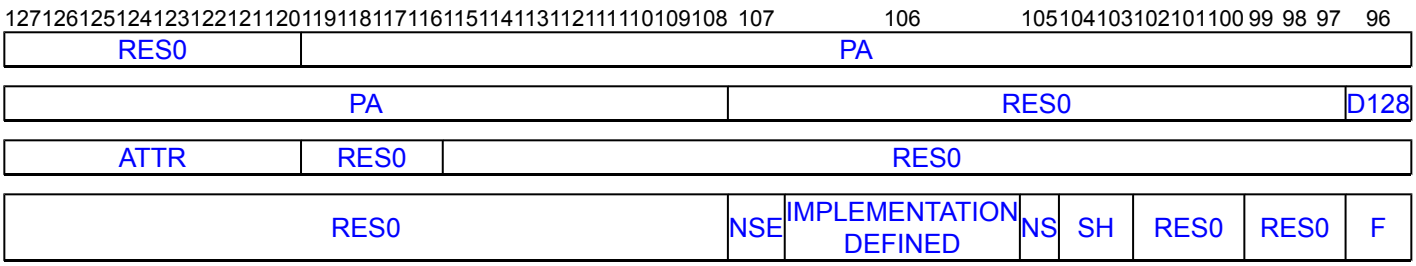
Attributes

PAR_EL1 is a:

- 128-bit register when FEAT_D128 is implemented, GetPAR_EL1_D128() == 1, and GetPAR_EL1_F() == 0
- 128-bit register when FEAT_D128 is implemented, GetPAR_EL1_D128() == 1, and GetPAR_EL1_F() == 1
- 128-bit register when FEAT_D128 is implemented, GetPAR_EL1_D128() == 0, and GetPAR_EL1_F() == 0
- 128-bit register when FEAT_D128 is implemented, GetPAR_EL1_D128() == 0, and GetPAR_EL1_F() == 1
- 64-bit register when FEAT_D128 is not implemented and GetPAR_EL1_F() == 0
- 64-bit register when FEAT_D128 is not implemented and GetPAR_EL1_F() == 1

Field descriptions

When FEAT_D128 is implemented, GetPAR_EL1_D128() == 1, and GetPAR_EL1_F() == 0:



This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR_EL1 can return a value that indicates the resulting attributes, rather than the values that appear in the Translation table descriptors. More precisely:

- The PAR_EL1.{ATTR, SH} fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the Translation table descriptors.
- See the PAR_EL1.NS bit description for constraints on the value it returns.

Bits [127:120]

Reserved, RES0.

PA, bits [119:76]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[55:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [75:65]

Reserved, RES0.

D128, bit [64]

Indicates if the PAR_EL1 uses the 128-bit format.

D128	Meaning
0b1	PAR_EL1 uses the 128-bit format. PAR_EL1[127:0] holds valid data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ATTR, bits [63:56]

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in MAIR_EL1, MAIR_EL2, and MAIR_EL3.

If FEAT_MTE_PERM is implemented and the instruction performed a stage 2 translation, the following additional encoding is defined:

ATTR	Meaning
0b11100000	Tagged NoTagAccess Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory.

Note

This encoding in MAIR_ELx is Reserved.

The value returned in this field can be the resulting attribute that is actually implemented by the implementation, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

Note

The attributes presented are consistent with the stages of translation applied in the address translation instruction. If the instruction performed a stage 1 translation only, the attributes are from the stage 1 translation. If the instruction performed a stage 1 and stage 2 translation, the attributes are from the combined stage 1 and stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52, 6:4]

Reserved, RES0.

Bits [51:12]

Reserved, RES0.

NSE, bit [11]

When FEAT_RME is implemented:

Reports the NSE attribute for a translation table descriptor from the EL3 translation regime.

For a description of the values derived by evaluating NS and NSE together, see PAR_EL1.NS.

For a result from a Secure, Non-secure, or Realm translation regime, this bit is unknown.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IMPLEMENTATION DEFINED, bit [10]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

When FEAT_RME is implemented:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime, a Realm translation regime, and the EL3 translation regime.

For a result from an EL3 translation regime, NS and NSE are evaluated together to report the physical address space:

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

For a result from an S1E1 or S1E0 operation on the Realm EL1&0 translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [8:7]

Shareability attribute, for the returned output address.

SH	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

The value 0b01 is reserved.

Note

This field returns the value 0b10 for:

- Any type of Device memory.
- Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:1]

Reserved, RES0.

F, bit [0]

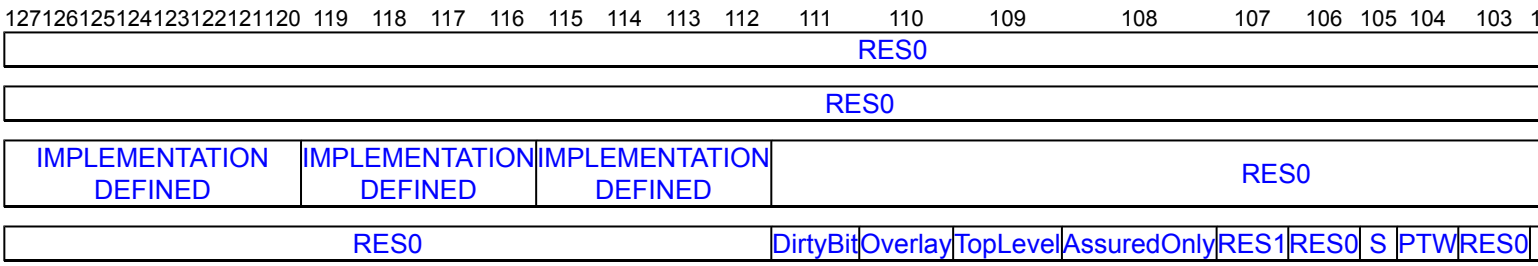
Indicates whether the instruction performed a successful address translation.

F	Meaning
0b0	Address translation completed successfully.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is implemented, GetPAR_EL1_D128() == 1, and GetPAR_EL1_F() == 1:



This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

Bits [127:65]

Reserved, RES0.

D128, bit [64]

Indicates if the PAR_EL1 uses the 128-bit format.

D128	Meaning
0b1	PAR_EL1 uses the 128-bit format. PAR_EL1[127:0] holds valid data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [63:56]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [55:52]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [51:48]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:16]

Reserved, RES0.

DirtyBit, bit [15]

When FEAT_S1PIE is implemented or FEAT_S2PIE is implemented:

DirtyBit flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation that is using Indirect Permissions, and dirty state is managed by software, then this field holds information about the fault.

DirtyBit	Meaning
0b0	The Permission Fault is not due to dirty state.
0b1	The Permission Fault is due to dirty state.

For any other fault or Access, this field is RES0.

Note

At stage 1, dirty state is indicated by the nDirty bit in Block and Page descriptors. At stage 2, dirty state is indicated by the Dirty bit in Block and Page descriptors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Overlay, bit [14]

When FEAT_S1POE is implemented or FEAT_S2POE is implemented:

Overlay flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation, then this field holds information about the fault.

Overlay	Meaning
0b0	The Data Abort is not due to Overlay Permissions.
0b1	The Data Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TopLevel, bit [13]

When FEAT_THE is implemented:

Fault due to TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [12]

When FEAT_THE is implemented:

AssuredOnly flag.

If PAR_EL1.S indicates a stage 2 fault, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	The Data Abort is not due to AssuredOnly.
0b1	The Data Abort is due to AssuredOnly.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES1.

Bit [10]

Reserved, RES0.

S, bit [9]

Indicates the translation stage at which the translation aborted:

S	Meaning
0b0	Translation aborted because of a fault in the stage 1 translation.
0b1	Translation aborted because of a fault in the stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PTW, bit [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

FST, bits [6:1]

Fault status code, as shown in the Data Abort ESR encoding.

FST	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	When FEAT_RAS is not implemented
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented

0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented
0b111101	Section Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	When FEAT_AA32EL1 is implemented
0b111110	Page Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	When FEAT_AA32EL1 is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

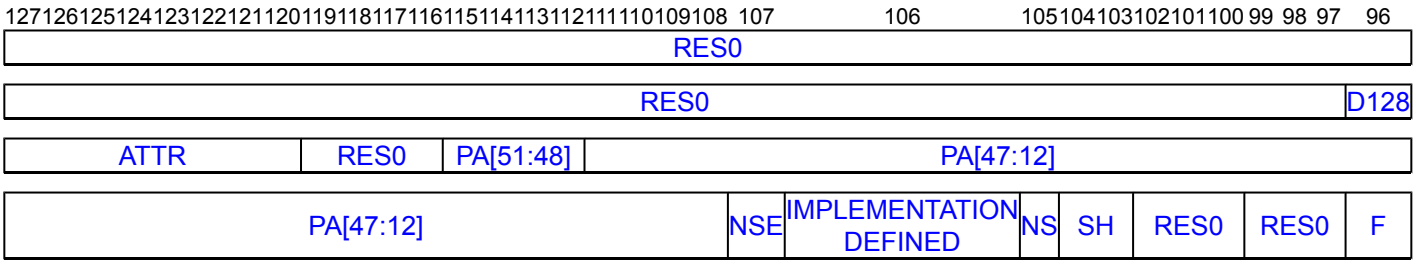
Indicates whether the instruction performed a successful address translation.

F	Meaning
0b1	Address translation aborted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is implemented, GetPAR_EL1_D128() == 0, and GetPAR_EL1_F() == 0:



This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR_EL1 can return a value that indicates the resulting attributes, rather than the values that appear in the Translation table descriptors. More precisely:

- The PAR_EL1.{ATTR, SH} fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the Translation table descriptors.
- See the PAR_EL1.NS bit description for constraints on the value it returns.

Bits [127:65]

Reserved, RES0.

D128, bit [64]

Indicates if the PAR_EL1 uses the 128-bit format.

D128	Meaning
0b0	PAR_EL1 uses the 64-bit format. PAR_EL1[63:0] holds valid data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ATTR, bits [63:56]

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in MAIR_EL1, MAIR_EL2, and MAIR_EL3.

If FEAT_MTE_PERM is implemented and the instruction performed a stage 2 translation, the following additional encoding is defined:

ATTR	Meaning
0b11100000	Tagged NoTagAccess Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory.

Note

This encoding in MAIR_ELx is Reserved.

The value returned in this field can be the resulting attribute that is actually implemented by the implementation, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

Note

The attributes presented are consistent with the stages of translation applied in the address translation instruction. If the instruction performed a stage 1 translation only, the attributes are from the stage 1 translation. If the instruction performed a stage 1 and stage 2 translation, the attributes are from the combined stage 1 and stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52, 6:4]

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

Extension to PA[47:12]. For more information, see PA[47:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[47:12], bits [47:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[47:12].

When FEAT_LPA is implemented and 52-bit addresses are in use, PA[51:48] forms the upper part of the address value. Otherwise, when 52-bit addresses are not in use, PA[51:48] is RES0.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [11]

When FEAT_RME is implemented:

Reports the NSE attribute for a translation table entry from the EL3 translation regime.

For a description of the values derived by evaluating NS and NSE together, see PAR_EL1.NS.

For a result from a Secure, Non-secure, or Realm translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IMPLEMENTATION DEFINED, bit [10]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

When FEAT_RME is implemented:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime, a Realm translation regime, and the EL3 translation regime.

For a result from an EL3 translation regime, NS and NSE are evaluated together to report the physical address space:

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

For a result from an S1E1 or S1E0 operation on the Realm EL1&0 translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [8:7]

Shareability attribute, for the returned output address.

SH	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

The value 0b01 is reserved.

Note

This field returns the value 0b10 for:

- Any type of Device memory.
- Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:1]

Reserved, RES0.

F, bit [0]

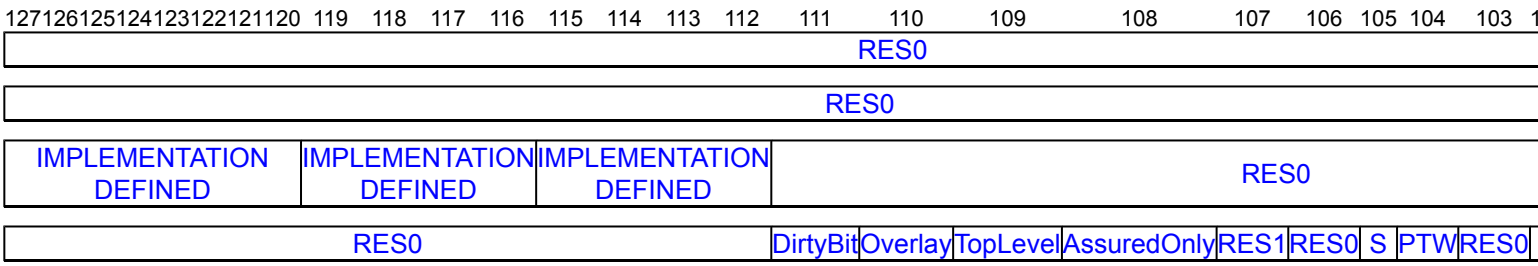
Indicates whether the instruction performed a successful address translation.

F	Meaning
0b0	Address translation completed successfully.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is implemented, GetPAR_EL1_D128() == 0, and GetPAR_EL1_F() == 1:



This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

Bits [127:65]

Reserved, RES0.

D128, bit [64]

Indicates if the PAR_EL1 uses the 128-bit format.

D128	Meaning
0b0	PAR_EL1 uses the 64-bit format. PAR_EL1[63:0] holds valid data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [63:56]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [55:52]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [51:48]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:16]

Reserved, RES0.

DirtyBit, bit [15]

When FEAT_S1PIE is implemented or FEAT_S2PIE is implemented:

DirtyBit flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation that is using Indirect Permissions, and dirty state is managed by software, then this field holds information about the fault.

DirtyBit	Meaning
0b0	The Permission Fault is not due to nDirty State or Dirty State.
0b1	The Permission Fault is due to nDirty State or Dirty State.

For any other fault or Access, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Overlay, bit [14]

When FEAT_S1POE is implemented or FEAT_S2POE is implemented:

Overlay flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation, then this field holds information about the fault.

Overlay	Meaning
0b0	The Data Abort is not due to Overlay Permissions.
0b1	The Data Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TopLevel, bit [13]

When FEAT_THE is implemented:

Fault due to TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [12]

When FEAT_THE is implemented:

AssuredOnly flag.

If PAR_EL1.S indicates a stage 2 fault, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	The Data Abort is not due to AssuredOnly.
0b1	The Data Abort is due to AssuredOnly.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES1.

Bit [10]

Reserved, RES0.

S, bit [9]

Indicates the translation stage at which the translation aborted:

S	Meaning
0b0	Translation aborted because of a fault in the stage 1 translation.
0b1	Translation aborted because of a fault in the stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PTW, bit [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

FST, bits [6:1]

Fault status code, as shown in the Data Abort exception ESR encoding.

FST	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	When FEAT_RAS is not implemented
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented

0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented
0b111101	Section Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	When FEAT_AA32EL1 is implemented
0b111110	Page Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	When FEAT_AA32EL1 is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

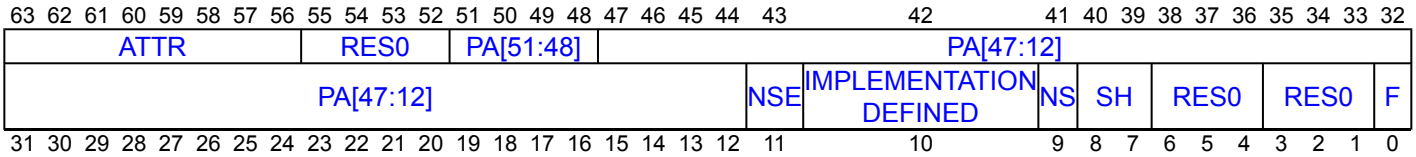
Indicates whether the instruction performed a successful address translation.

F	Meaning
0b1	Address translation aborted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is not implemented and GetPAR_EL1_F() == 0:



This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR_EL1 can return a value that indicates the resulting attributes, rather than the values that appear in the Translation table descriptors. More precisely:

- The PAR_EL1.{ATTR, SH} fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the Translation table descriptors.
- See the PAR_EL1.NS bit description for constraints on the value it returns.

ATTR, bits [63:56]

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in MAIR_EL1, MAIR_EL2, and MAIR_EL3.

If FEAT_MTE_PERM is implemented and the instruction performed a stage 2 translation, the following additional encoding is defined:

ATTR	Meaning
0b11100000	Tagged NoTagAccess Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory.

Note

This encoding in MAIR_ELx is Reserved.

The value returned in this field can be the resulting attribute that is actually implemented by the implementation, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

Note

The attributes presented are consistent with the stages of translation applied in the address translation instruction. If the instruction performed a stage 1 translation only, the attributes are from the stage 1 translation. If the instruction performed a stage 1 and stage 2 translation, the attributes are from the combined stage 1 and stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52, 6:4]

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

Extension to PA[47:12]. For more information, see PA[47:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[47:12], bits [47:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[47:12].

When FEAT_LPA is implemented and 52-bit addresses are in use, PA[51:48] forms the upper part of the address value. Otherwise, when 52-bit addresses are not in use, PA[51:48] is RES0.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [11]

When FEAT_RME is implemented:

Reports the NSE attribute for a translation table entry from the EL3 translation regime.

For a description of the values derived by evaluating NS and NSE together, see PAR_EL1.NS.

For a result from a Secure, Non-secure, or Realm translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IMPLEMENTATION DEFINED, bit [10]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

When FEAT_RME is implemented:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime, a Realm translation regime, and the EL3 translation regime.

For a result from an EL3 translation regime, NS and NSE are evaluated together to report the physical address space:

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

For a result from an S1E1 or S1E0 operation on the Realm EL1&0 translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [8:7]

Shareability attribute, for the returned output address.

SH	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

The value 0b01 is reserved.

Note

This field returns the value 0b10 for:

- Any type of Device memory.
- Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:1]

Reserved, RES0.

F, bit [0]

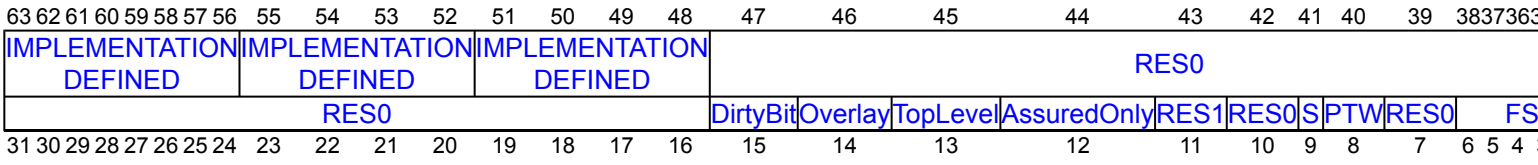
Indicates whether the instruction performed a successful address translation.

F	Meaning
0b0	Address translation completed successfully.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is not implemented and GetPAR_EL1_F() == 1:



This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

IMPLEMENTATION DEFINED, bits [63:56]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [55:52]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [51:48]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:16]

Reserved, RES0.

DirtyBit, bit [15]

When FEAT_S1PIE is implemented or FEAT_S2PIE is implemented:

DirtyBit flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation that is using Indirect Permissions, and dirty state is managed by software, then this field holds information about the fault.

DirtyBit	Meaning
0b0	The Permission Fault is not due to nDirty State or Dirty State.
0b1	The Permission Fault is due to nDirty State or Dirty State.

For any other fault or Access, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Overlay, bit [14]

When FEAT_S1POE is implemented or FEAT_S2POE is implemented:

Overlay flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation, then this field holds information about the fault.

Overlay	Meaning
0b0	The Data Abort is not due to Overlay Permissions.
0b1	The Data Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TopLevel, bit [13]

When FEAT_THE is implemented:

Fault due to TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [12]

When FEAT_THE is implemented:

AssuredOnly flag.

If PAR_EL1.S indicates a stage 2 fault, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	The Data Abort is not due to AssuredOnly.
0b1	The Data Abort is due to AssuredOnly.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES1.

Bit [10]

Reserved, RES0.

S, bit [9]

Indicates the translation stage at which the translation aborted:

S	Meaning
0b0	Translation aborted because of a fault in the stage 1 translation.
0b1	Translation aborted because of a fault in the stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PTW, bit [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

FST, bits [6:1]

Fault status code, as shown in the Data Abort exception ESR encoding.

FST	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	When FEAT_RAS is not implemented
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented

0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented
0b111101	Section Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	When FEAT_AA32EL1 is implemented
0b111110	Page Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	When FEAT_AA32EL1 is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

F	Meaning
0b1	Address translation aborted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0111	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = PAR_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    X[t, 64] = PAR_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PAR_EL1<63:0>;
```

```
MSR PAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0111	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGWTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        PAR_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    PAR_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    PAR_EL1<63:0> = X[t, 64];
```

When FEAT_D128 is implemented

```
MRRS <Xt>, <Xt+1>, PAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0111	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = PAR_EL1<127:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = PAR_EL1<127:0>;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = PAR_EL1<127:0>;
```

When FEAT_D128 is implemented

```
MSRR PAR_EL1, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0111	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        PAR_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        PAR_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    PAR_EL1<127:0> = X[t, t2, 128];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PFAR_EL1, Physical Fault Address Register (EL1)

The PFAR_EL1 characteristics are:

Purpose

Records the faulting physical address for a synchronous External abort, or SError exception taken to EL1.

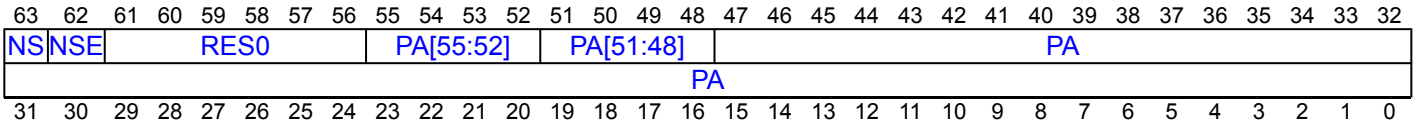
Configuration

This register is present only when FEAT_PFAR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PFAR_EL1 are UNDEFINED.

Attributes

PFAR_EL1 is a 64-bit register.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

Together with PFAR_EL1.NSE, reports the physical address space of the access that triggered the exception.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Reserved.
0b1	0b1	Realm.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When EL3 is implemented or FEAT_Secure is implemented:

Non-secure. Reports the physical address space of the access that triggered the exception.

NS	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSE, bit [62]

When FEAT_RME is implemented:

Together with PFAR_EL1.NS, reports the physical address space of the access that triggered the exception.

For a description of the values derived by evaluating NS and NSE together, see MFAR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [61:56]

Reserved, RES0.

PA[55:52], bits [55:52]

When FEAT_D128 is implemented:

When FEAT_D128 is implemented, extension to PFAR_EL1.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

When FEAT_LPA is implemented, extension to PFAR_EL1.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA, bits [47:0]

Physical Address. Bits [47:0] of the aborting physical address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The recorded address can be any address within the same naturally-aligned fault granule as the faulting physical address, where the size of the fault granule is IMPLEMENTATION DEFINED and no larger than the larger than:

- The size of the range of values permitted to be recorded in FAR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PFAR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name PFAR_EL1 or PFAR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

PFAR_EL1 is not valid and reads UNKNOWN if ESR_EL1.PFV is recorded as 0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PFAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nPFAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x2D0];
    else
        X[t, 64] = PFAR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PFAR_EL2;
    else
        X[t, 64] = PFAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PFAR_EL1;
```

```
MSR PFAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nPFAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x2D0] = X[t, 64];
    else
        PFAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PFAR_EL2 = X[t, 64];
    else
        PFAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PFAR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, PFAR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x2D0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PFAR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = PFAR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR PFAR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x2D0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PFAR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        PFAR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PFAR_EL2, Physical Fault Address Register (EL2)

The PFAR_EL2 characteristics are:

Purpose

Records the faulting physical address for a synchronous External abort, or SError exception taken to EL2.

Configuration

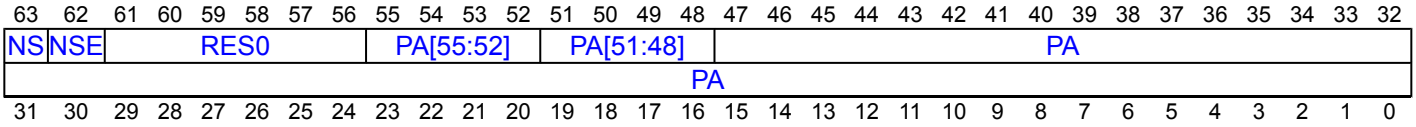
This register is present only when FEAT_PFAR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PFAR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

PFAR_EL2 is a 64-bit register.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

Together with PFAR_EL2.NSE, reports the physical address space of the access that triggered the exception.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Reserved.
0b1	0b1	Realm.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When EL3 is implemented or FEAT_Secure is implemented:

Non-secure. Reports the physical address space of the access that triggered the exception.

NS	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSE, bit [62]

When FEAT_RME is implemented:

Together with PFAR_EL2.NS, reports the physical address space of the access that triggered the exception.

For a description of the values derived by evaluating NS and NSE together, see MFAR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [61:56]

Reserved, RES0.

PA[55:52], bits [55:52]

When FEAT_D128 is implemented:

When FEAT_D128 is implemented, extension to PFAR_EL2.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

When FEAT_LPA is implemented, extension to PFAR_EL2.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA, bits [47:0]

Physical Address. Bits [47:0] of the aborting physical address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The recorded address can be any address within the same naturally-aligned fault granule as the faulting physical address, where the size of the fault granule is IMPLEMENTATION DEFINED and no larger than the larger than:

- The size of the range of values permitted to be recorded in FAR_EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PFAR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name PFAR_EL2 or PFAR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

PFAR_EL2 is not valid and reads UNKNOWN if ESR_EL2.PFV is recorded as 0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PFAR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PFAR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PFAR_EL2;
```

```
MSR PFAR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PFAR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PFAR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PIR_EL1, Permission Indirection Register 1 (EL1)

The PIR_EL1 characteristics are:

Purpose

Stage 1 Permission Indirection Register for privileged access of the EL1&0 translation regime.

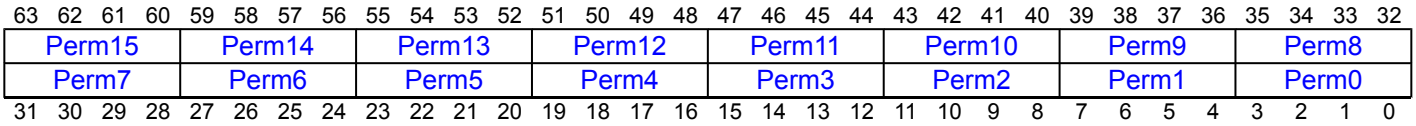
Configuration

This register is present only when FEAT_S1PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PIR_EL1 are UNDEFINED.

Attributes

PIR_EL1 is a 64-bit register.

Field descriptions



Perm<m>, bits [4m+3:4m], for m = 15 to 0

Represents stage 1 Base Permissions.

Perm<m>	Meaning
0b0000	No access. Overlay applied.
0b0001	Read. Overlay applied.
0b0010	Execute. Overlay applied.
0b0011	Read and Execute. Overlay applied.
0b0100	Reserved - treated as No access. Overlay applied.
0b0101	Read and Write. Overlay applied.
0b0110	Read, Write, and Execute. Overlay applied. WXN control applied.
0b0111	Read, Write, and Execute. Overlay applied.
0b1000	Read. Overlay not applied.
0b1001	Read, GCS Read, and GCS Write. Overlay not applied.
0b1010	Read and Execute. Overlay not applied.
0b1011	Reserved - treated as No access. Overlay not applied.
0b1100	Read and Write. Overlay not applied.
0b1101	Reserved - treated as No access. Overlay not applied.
0b1110	Read, Write, and Execute. Overlay not applied.
0b1111	Reserved - treated as No access. Overlay not applied.

This field is permitted to be cached in a TLB.

When stage 1 Indirect Permission mechanism is disabled, this register is ignored.

Unless otherwise specified, the WXN control is not applied.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PIR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name PIR_EL1 or PIR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PIR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nPIR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x2A0];
    else
        X[t, 64] = PIR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PIR_EL2;
    else
        X[t, 64] = PIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIR_EL1;
```

```
MSR PIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nPIR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x2A0] = X[t, 64];
    else
        PIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PIR_EL2 = X[t, 64];
    else
        PIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, PIR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x2A0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PIR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = PIR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR PIR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x2A0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PIR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        PIR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PIR_EL2, Permission Indirection Register 2 (EL2)

The PIR_EL2 characteristics are:

Purpose

Stage 1 Permission Indirection Register for privileged access of the EL2 or EL2&0 translation regime.

Configuration

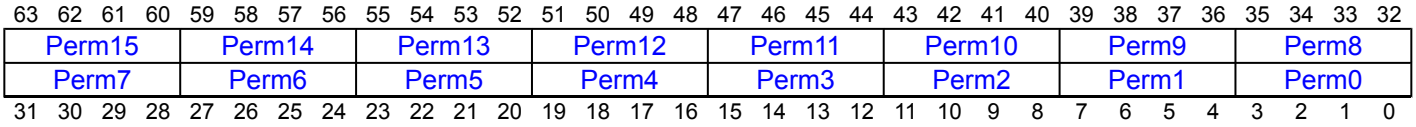
This register is present only when FEAT_S1PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PIR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

PIR_EL2 is a 64-bit register.

Field descriptions



Perm<m>, bits [4m+3:4m], for m = 15 to 0

Represents stage 1 Base Permissions.

Perm<m>	Meaning
0b0000	No access. Overlay applied.
0b0001	Read. Overlay applied.
0b0010	Execute. Overlay applied.
0b0011	Read and Execute. Overlay applied.
0b0100	Reserved - treated as No access. Overlay applied.
0b0101	Read and Write. Overlay applied.
0b0110	Read, Write, and Execute. Overlay applied. WXN control applied.
0b0111	Read, Write, and Execute. Overlay applied.
0b1000	Read. Overlay not applied.
0b1001	Read, GCS Read, and GCS Write. Overlay not applied.
0b1010	Read and Execute. Overlay not applied.
0b1011	Reserved - treated as No access. Overlay not applied.
0b1100	Read and Write. Overlay not applied.
0b1101	Reserved - treated as No access. Overlay not applied.
0b1110	Read, Write, and Execute. Overlay not applied.
0b1111	Reserved - treated as No access. Overlay not applied.

This field is permitted to be cached in a TLB.

When stage 1 Indirect Permission mechanism is disabled, this register is ignored.

Unless otherwise specified, the WXN control is not applied.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PIR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name PIR_EL2 or PIR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PIR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PIR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIR_EL2;
```

```
MSR PIR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PIR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIR_EL2 = X[t, 64];
```

MRS <Xt>, PIR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nPIR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x2A0];
    else
        X[t, 64] = PIR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PIR_EL2;
    else
        X[t, 64] = PIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIR_EL1;
```

MSR PIR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nPIR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x2A0] = X[t, 64];
    else
        PIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PIR_EL2 = X[t, 64];
    else
        PIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PIR_EL3, Permission Indirection Register 3 (EL3)

The PIR_EL3 characteristics are:

Purpose

Stage 1 Permission Indirection Register for privileged access of the EL3 translation regime.

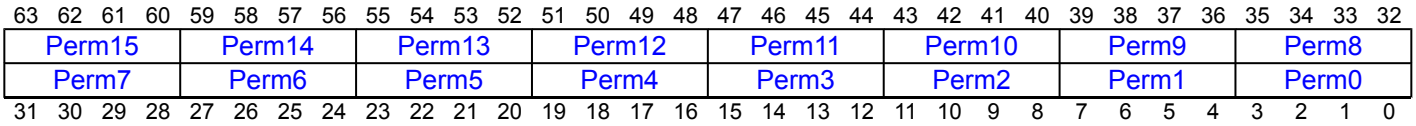
Configuration

This register is present only when FEAT_S1PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PIR_EL3 are UNDEFINED.

Attributes

PIR_EL3 is a 64-bit register.

Field descriptions



Perm<m>, bits [4m+3:4m], for m = 15 to 0

Represents stage 1 Base Permissions.

Perm<m>	Meaning
0b0000	No access. Overlay applied.
0b0001	Read. Overlay applied.
0b0010	Execute. Overlay applied.
0b0011	Read and Execute. Overlay applied.
0b0100	Reserved - treated as No access. Overlay applied.
0b0101	Read and Write. Overlay applied.
0b0110	Read, Write, and Execute. Overlay applied. WXN control applied.
0b0111	Read, Write, and Execute. Overlay applied.
0b1000	Read. Overlay not applied.
0b1001	Read, GCS Read, and GCS Write. Overlay not applied.
0b1010	Read and Execute. Overlay not applied.
0b1011	Reserved - treated as No access. Overlay not applied.
0b1100	Read and Write. Overlay not applied.
0b1101	Reserved - treated as No access. Overlay not applied.
0b1110	Read, Write, and Execute. Overlay not applied.
0b1111	Reserved - treated as No access. Overlay not applied.

This field is permitted to be cached in a TLB.

When stage 1 Indirect Permission mechanism is disabled, this register is ignored.

Unless otherwise specified, the WXN control is not applied.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PIR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PIR_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIR_EL3;
```

MSR PIR_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.PIR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PIR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PIRE0_EL1, Permission Indirection Register 0 (EL1)

The PIRE0_EL1 characteristics are:

Purpose

Stage 1 Permission Indirection Register for unprivileged access of the EL1&0 translation regime.

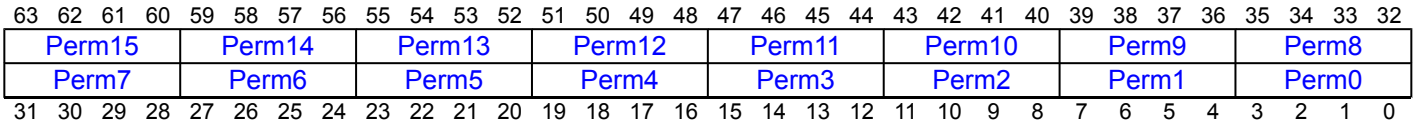
Configuration

This register is present only when FEAT_S1PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PIRE0_EL1 are UNDEFINED.

Attributes

PIRE0_EL1 is a 64-bit register.

Field descriptions



Perm<m>, bits [4m+3:4m], for m = 15 to 0

Represents stage 1 Base Permissions.

Perm<m>	Meaning
0b0000	No access. Overlay applied.
0b0001	Read. Overlay applied.
0b0010	Execute. Overlay applied.
0b0011	Read and Execute. Overlay applied.
0b0100	Reserved - treated as No access. Overlay applied.
0b0101	Read and Write. Overlay applied.
0b0110	Read, Write, and Execute. Overlay applied. WXN control applied.
0b0111	Read, Write, and Execute. Overlay applied.
0b1000	Read. Overlay not applied.
0b1001	Read, GCS Read, and GCS Write. Overlay not applied.
0b1010	Read and Execute. Overlay not applied.
0b1011	Reserved - treated as No access. Overlay not applied.
0b1100	Read and Write. Overlay not applied.
0b1101	Reserved - treated as No access. Overlay not applied.
0b1110	Read, Write, and Execute. Overlay not applied.
0b1111	Reserved - treated as No access. Overlay not applied.

This field is permitted to be cached in a TLB.

When stage 1 Indirect Permission mechanism is disabled, this register is ignored.

Unless otherwise specified, the WXN control is not applied.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PIRE0_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name PIRE0_EL1 or PIRE0_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PIRE0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nPIRE0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x290];
    else
        X[t, 64] = PIRE0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PIRE0_EL2;
    else
        X[t, 64] = PIRE0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIRE0_EL1;
```

```
MSR PIRE0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nPIRE0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x290] = X[t, 64];
    else
        PIRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PIRE0_EL2 = X[t, 64];
    else
        PIRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIRE0_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, PIRE0_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x290];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PIRE0_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = PIRE0_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR PIRE0_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x290] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PIRE0_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        PIRE0_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PIRE0_EL2, Permission Indirection Register 0 (EL2)

The PIRE0_EL2 characteristics are:

Purpose

Stage 1 Permission Indirection Register for unprivileged access of the EL2&0 translation regime.

Configuration

This register is present only when FEAT_S1PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PIRE0_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

PIRE0_EL2 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Perm15				Perm14				Perm13				Perm12				Perm11				Perm10				Perm9				Perm8			
Perm7				Perm6				Perm5				Perm4				Perm3				Perm2				Perm1				Perm0			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Perm<m>, bits [4m+3:4m], for m = 15 to 0

Represents stage 1 Base Permissions.

Perm<m>	Meaning
0b0000	No access. Overlay applied.
0b0001	Read. Overlay applied.
0b0010	Execute. Overlay applied.
0b0011	Read and Execute. Overlay applied.
0b0100	Reserved - treated as No access. Overlay applied.
0b0101	Read and Write. Overlay applied.
0b0110	Read, Write, and Execute. Overlay applied. WXN control applied.
0b0111	Read, Write, and Execute. Overlay applied.
0b1000	Read. Overlay not applied.
0b1001	Read, GCS Read, and GCS Write. Overlay not applied.
0b1010	Read and Execute. Overlay not applied.
0b1011	Reserved - treated as No access. Overlay not applied.
0b1100	Read and Write. Overlay not applied.
0b1101	Reserved - treated as No access. Overlay not applied.
0b1110	Read, Write, and Execute. Overlay not applied.
0b1111	Reserved - treated as No access. Overlay not applied.

This field is permitted to be cached in a TLB.

When stage 1 Indirect Permission mechanism is disabled, this register is ignored.

Unless otherwise specified, the WXN control is not applied.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PIRE0_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name PIRE0_EL2 or PIRE0_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PIRE0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PIRE0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIRE0_EL2;
```

```
MSR PIRE0_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PIRE0_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIRE0_EL2 = X[t, 64];
```

MRS <Xt>, PIRE0_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nPIRE0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x290];
    else
        X[t, 64] = PIRE0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PIRE0_EL2;
    else
        X[t, 64] = PIRE0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIRE0_EL1;
```

MSR PIRE0_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nPIRE0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x290] = X[t, 64];
    else
        PIRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PIRE0_EL2 = X[t, 64];
    else
        PIRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIRE0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PM, Profiling Exception Mask

The PM characteristics are:

Purpose

Allows access to the Profiling exception Mask bit.

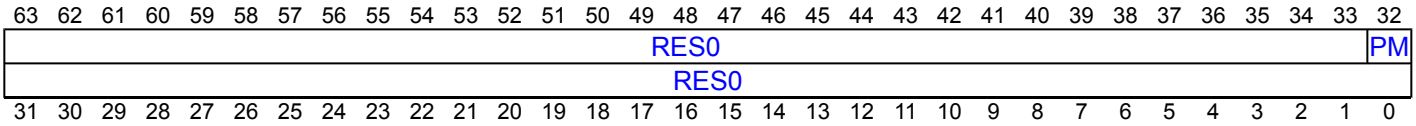
Configuration

This register is present only when FEAT_EBEP is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PM are UNDEFINED.

Attributes

PM is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

PM, bit [32]

PMU Exception Mask.

PM	Meaning
0b0	Does not cause the Profiling exception to be masked.
0b1	Causes the Profiling exception to be masked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

Accessing PM

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PM
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_EBEP) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    X[t, 64] = Zeros(31):PSTATE.PM:Zeros(32);
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(31):PSTATE.PM:Zeros(32);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(31):PSTATE.PM:Zeros(32);
```

MSR PM, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_EBEP) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    PSTATE.PM = X[t, 64]<32>;
elsif PSTATE.EL == EL2 then
    PSTATE.PM = X[t, 64]<32>;
elsif PSTATE.EL == EL3 then
    PSTATE.PM = X[t, 64]<32>;
```

MSR PM, #<imm>

op0	op1	CRn	CRm	op2
0b00	0b001	0b0100	0b001x	0b000

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBIDR_EL1, Profiling Buffer ID Register

The PMBIDR_EL1 characteristics are:

Purpose

Provides information to software as to whether the buffer can be programmed at the current Exception level.

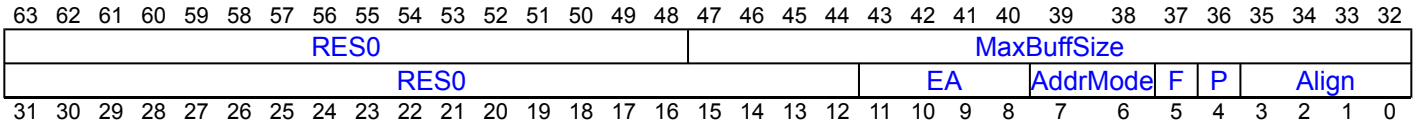
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBIDR_EL1 are UNDEFINED.

Attributes

PMBIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:48]

Reserved, RES0.

MaxBuffSize, bits [47:32]

Maximum supported Profiling Buffer size. Reserved for software use.

The only permitted value is `0x0000`, indicating there is no limit to the maximum buffer size.

Note

Permitted values relate to the values an implementation is permitted to set this field to. A hypervisor might trap accesses to this register and use other values to describe limitations of its virtualization support to a guest operating system, as follows:

- MaxBuffSize bits[8:0] encodes a mantissa value, M.
- MaxBuffSize bits[13:9] encodes an exponent value, E.
- MaxBuffSize bits[15:14] are reserved.

The maximum buffer size, in bytes, is expressed using the following function:

if IsZero(E) then UInt(M:Zeros(12)) else UInt('1':M:Zeros(UInt(E)+11))

For example:

- A value of `0x0001` means a maximum buffer size of 4KB.
- A value of `0x3FFF` means a maximum buffer size of 4092TB.

Reads as `0x0000`.

Access to this field is **RO**.

Bits [31:12]

Reserved, RES0.

EA, bits [11:8]

External Abort handling. Describes how the PE manages External aborts on writes made by the Statistical Profiling Unit to the Profiling Buffer.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EA	Meaning
0b0000	Not described.
0b0001	The PE ignores External aborts on writes made by the Statistical Profiling Unit.
0b0010	An External abort on a write made by the Statistical Profiling Unit generates an asynchronous SError exception at the PE.

All other values are reserved.

From Armv8.8, the value 0b0000 is not permitted.

PMBIDR_EL1.EA describes only External aborts generated by the write to memory. External aborts on a translation table walk made by the Statistical Profiling Unit generate Profiling Buffer management events reported as MMU faults using PMBSR_ELx.

Access to this field is **RO**.

AddrMode, bits [7:6]

When FEAT_SPE_nVM is implemented:

Address Modes. Describes the addressing modes available for the Profiling Buffer.

AddrMode	Meaning
0b00	Only virtual address mode is supported.
0b01	Virtual and physical address modes are supported.
0b11	Reserved for software use under virtualization, to show that only physical address mode is supported.

Other values are reserved.

If the Effective value of PMSCR_EL2.EnVM is 1 and the value returned for PMBIDR_EL1.P is 0, then this field reads as 0b01. Otherwise, this field reads as 0b00.

Note

A hypervisor might trap accesses to this register to describe limitations of its virtualization support to a guest operating system.

Otherwise:

Reserved, RES0.

F, bit [5]

Flag updates. Describes how address translations performed by the Statistical Profiling Unit manage the Access flag and dirty state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F	Meaning
0b0	Hardware management of the Access flag and dirty state for accesses made by the Statistical Profiling Unit is always disabled for all translation stages.
0b1	Hardware management of the Access flag and dirty state for accesses made by the Statistical Profiling Unit is controlled in the same way as explicit memory accesses in the Profiling Buffer owning translation regime.

Note

If hardware management of the Access flag is disabled for a stage of translation, an access to a Page or Block with the Access flag bit not set in the descriptor will generate an Access Flag fault.

If hardware management of the dirty state is disabled for a stage of translation, an access to a Page or Block will ignore the Dirty Bit Modifier in the descriptor and might generate a Permission fault, depending on the values of the access permission bits in the descriptor.

From Armv8.8, the value 0 is not permitted.

Access to this field is **RO**.

P, bit [4]

Programming not allowed. When read at EL3, this field reads as zero. Otherwise, indicates that the Profiling Buffer owning Exception level is a higher Exception level or the Profiling Buffer owning Security state is not the current Security state.

P	Meaning
0b0	Programming is allowed.
0b1	Programming not allowed.

The value read from this field depends on the current Exception level and the Effective values of MDCR_EL3.NSPB, MDCR_EL3.NSPBE, and MDCR_EL2.E2PB:

- If EL3 is implemented, MDCR_EL3.NSPB is 0b0x, and either FEAT_RME is not implemented, or Secure state is implemented and MDCR_EL3.NSPBE is 0, then this field reads as one from:
  - Non-secure EL1 and Non-secure EL2.
  - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
  - If Secure EL2 is implemented and enabled, and MDCR_EL2.E2PB is 0b00, Secure EL1.
- If EL3 is implemented, MDCR_EL3.NSPB is 0b1x and either FEAT_RME is not implemented or MDCR_EL3.NSPBE is 0, then this field reads as one from:
  - If Secure state is implemented, Secure EL1.
  - If Secure EL2 is implemented, Secure EL2.
  - If EL2 is implemented and MDCR_EL2.E2PB is 0b00, Non-secure EL1.
  - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
- If FEAT_RME is implemented, and MDCR_EL3.{NSPB, NSPBE} is {0b1x, 1}, then this field reads as one from:
  - Non-secure EL1 and Non-secure EL2.
  - If Secure state is implemented, Secure EL1 and Secure EL2.
  - If MDCR_EL2.E2PB is 0b00, Realm EL1.
- If EL3 is not implemented, EL2 is implemented, and MDCR_EL2.E2PB is 0b00, then this field reads as one from EL1.

Otherwise, this field reads as zero.

Align, bits [3:0]

Defines the minimum alignment constraint for writes to PMBPTR_EL1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Align	Meaning
0b0000	Byte.
0b0001	Halfword.
0b0010	Word.
0b0011	Doubleword.
0b0100	16 bytes.
0b0101	32 bytes.
0b0110	64 bytes.
0b0111	128 bytes.
0b1000	256 bytes.
0b1001	512 bytes.
0b1010	1KB.
0b1011	2KB.

All other values are reserved.

For more information, see 'Restrictions on the current write pointer'.

If this field is nonzero, then every record is a multiple of this size.

Access to this field is **RO**.

Accessing PMBIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b111

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMBIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = PMBIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = PMBIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBLIMITR_EL1, Profiling Buffer Limit Address Register

The PMBLIMITR_EL1 characteristics are:

Purpose

Defines the upper limit for the profiling buffer, and enables the profiling buffer

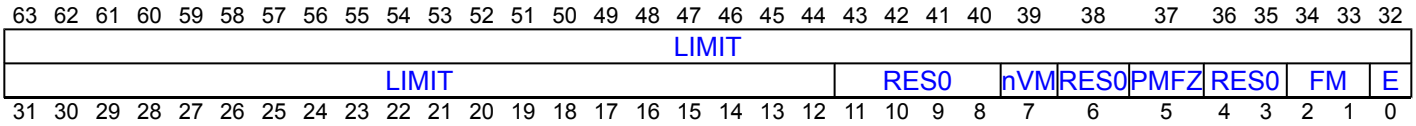
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBLIMITR_EL1 are UNDEFINED.

Attributes

PMBLIMITR_EL1 is a 64-bit register.

Field descriptions



LIMIT, bits [63:12]

Limit address. PMBLIMITR_EL1.LIMIT:Zeros(12) is the address of the first byte in memory after the last byte in the profiling buffer. If the smallest implemented translation granule is not 4KB, then bits[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value, Log_2(smallest implemented translation granule).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:8]

Reserved, RES0.

nVM, bit [7]

When FEAT_SPE_nVM is implemented:

Address mode.

nVM	Meaning
0b0	The Profiling Buffer pointers are virtual addresses.
0b1	The Profiling Buffer pointers are: • Physical address in the owning security state if the owning translation regime has no stage 2 translation. • Intermediate physical addresses in the owning security state if the owning translation regime has stage 2 translations.

If the Effective value of PMSCR_EL2.EnVM is 0, then the PE ignores the value of this field, and the Profiling Buffer pointers are always virtual addresses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RW**.

Otherwise:

Reserved, RES0.

Bit [6]

Reserved, RES0.

PMFZ, bit [5]

When FEAT_SPEv1p2 is implemented:

Freeze PMU on SPE event. Stop PMU event counters when PMBSR_EL1.S == 1.

PMFZ	Meaning
0b0	Do not freeze PMU event counters on Statistical Profiling Buffer Management event.
0b1	Freeze PMU event counters on Statistical Profiling Buffer Management event.

The PMU event counters affected by this control is controlled by PMCR_EL0.FZS and, if EL2 is implemented, MDCR_EL2.HPMFZS. See the descriptions of these control bits for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

FM, bits [2:1]

Fill mode.

FM	Meaning	Applies when
0b00	Fill mode. Stop collection and raise maintenance interrupt on buffer fill.	
0b10	Discard mode. All output is discarded.	When FEAT_SPEv1p2 is implemented

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Profiling Buffer enable

E	Meaning
0b0	All output is discarded.
0b1	Profiling buffer enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing PMBLIMITR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBLIMITR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMBLIMITR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x800];
    else
        X[t, 64] = PMBLIMITR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMBLIMITR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBLIMITR_EL1;
```

```
MSR PMBLIMITR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMBLIMITR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x800] = X[t, 64];
    else
        PMBLIMITR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBLIMITR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBLIMITR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBMAR_EL1, Profiling Buffer Memory Attribute Register

The PMBMAR_EL1 characteristics are:

Purpose

Controls Statistical Profiling Unit accesses to memory.

If the Profiling Buffer pointers specify virtual addresses, the address properties are defined by the translation tables and this register is ignored.

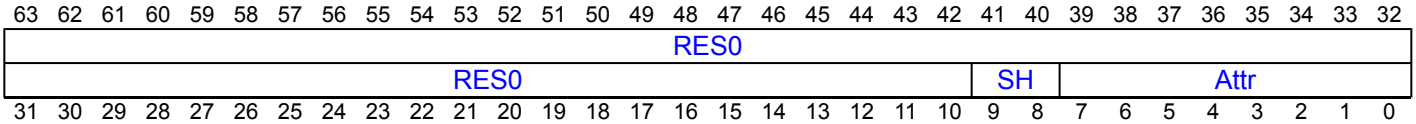
Configuration

This register is present only when FEAT_SPE_nVM is implemented. Otherwise, direct accesses to PMBMAR_EL1 are UNDEFINED.

Attributes

PMBMAR_EL1 is a 64-bit register.

Field descriptions



Bits [63:10]

Reserved, RES0.

SH, bits [9:8]

Profiling Buffer shareability domain. Defines the shareability domain for Normal memory used by the Profiling Buffer.

SH	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when PMBMAR_EL1.Attr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Attr, bits [7:0]

Profiling Buffer memory type and attributes. Defines the memory type and, for Normal memory, the cacheability attributes, for memory addressed by the Profiling Buffer.

The encoding of this field is the same as that of a MAIR_ELx.Attr<n> field, as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of 'dd' for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise,UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise,UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMBMAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBMAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b101

```
if !IsFeatureImplemented(FEAT_SPE_nVM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS4 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMBMAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS4 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMBMAR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS4 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS4 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMBMAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBMAR_EL1;
```

```
MSR PMBMAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b101

```
if !IsFeatureImplemented(FEAT_SPE_nVM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS4 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nPMBMAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS4 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBMAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS4 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS4 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBMAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBMAR_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBPTR_EL1, Profiling Buffer Write Pointer Register

The PMBPTR_EL1 characteristics are:

Purpose

Defines the current write pointer for the profiling buffer.

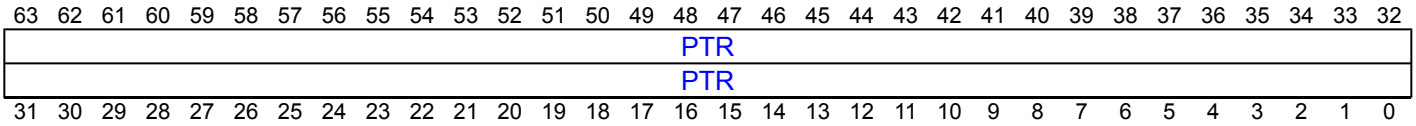
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBPTR_EL1 are UNDEFINED.

Attributes

PMBPTR_EL1 is a 64-bit register.

Field descriptions



PTR, bits [63:0]

Current write address. Defines the virtual address of the next entry to be written to the buffer.

If PMBIDR_EL1.Align is not zero, then it is IMPLEMENTATION DEFINED whether bits [M-1:0] are RES0 or read/write, where M is an integer between 1 and PMBIDR_EL1.Align inclusive.

The architecture places restrictions on the values software can write to the pointer when the SPU is not in Discard mode. For more information see 'Restrictions on the current write pointer'.

On a management interrupt, PMBPTR_EL1 is frozen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMBPTR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBPTR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b001

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMBPTR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x810];
    else
        X[t, 64] = PMBPTR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMBPTR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBPTR_EL1;
```

```
MSR PMBPTR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b001

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMBPTR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x810] = X[t, 64];
    else
        PMBPTR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBPTR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBPTR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBSR_EL1, Profiling Buffer Status/syndrome Register (EL1)

The PMBSR_EL1 characteristics are:

Purpose

Provides syndrome information to software for a Profiling Buffer management event.

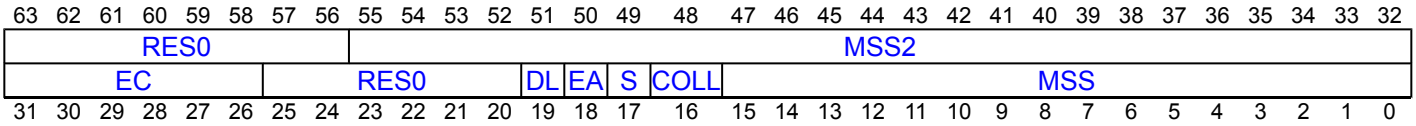
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBSR_EL1 are UNDEFINED.

Attributes

PMBSR_EL1 is a 64-bit register.

Field descriptions



Bits [63:56]

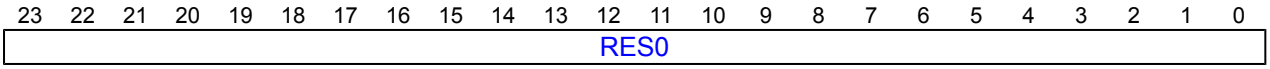
Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

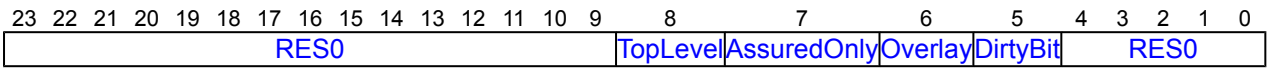
MSS2 encoding for other Profiling Buffer management events



Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer



Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, PMBSR_EL1.EC == 0b100101, and GetPMBSR_EL1_FSC() IN {0b0011xx}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetPMBSR_EL1_FSC() IN {0b0011xx}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetPMBSR_EL1_FSC() IN {0b0011xx}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

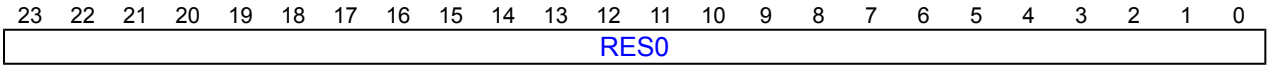
Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

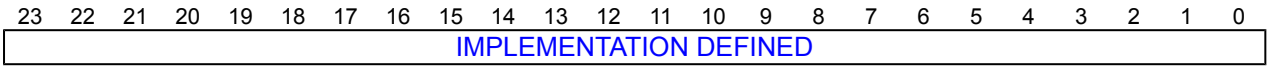
MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer



Bits [23:0]

Reserved, RES0.

MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason



IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the Profiling Buffer management event.

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other Profiling Buffer management event. All Profiling Buffer management events other than those described by the other defined Event class codes.	MSS encoding for other Profiling Buffer management events	MSS2 encoding for other Profiling Buffer management events	
0b011110	Granule Protection Check fault on write to Profiling Buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to Profiling Buffer	MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer	When FEAT_RME is implemented
0b011111	Profiling Buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	
0b100101	Stage 2 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:20]

Reserved, RES0.

DL, bit [19]

Partial record lost. Following a buffer management event other than an asynchronous External abort, indicates whether the last record written to the Profiling Buffer is complete.

DL	Meaning
0b0	PMBPTR_EL1 points to the first byte after the last complete record written to the Profiling Buffer.
0b1	Part of a record was lost because of a buffer management event or synchronous External abort. PMBPTR_EL1 might not point to the first byte after the last complete record written to the buffer, and so restarting collection might result in a data record stream that software cannot parse.

When the buffer management event was because of an asynchronous External abort, this bit is set to 1 and software must not assume that any valid data has been written to the Profiling Buffer.

This bit is RES0 if the PE never sets this bit as a result of a buffer management event caused by an asynchronous External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [18]

External abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Statistical Profiling Unit.

This bit is RES0 if the PE never sets this bit as the result of an External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [17]

Service. Indicates that a Profiling Buffer management event has been recorded.

S	Meaning
0b0	No Profiling Buffer management event for EL1 has been recorded.
0b1	A Profiling Buffer management event for EL1 has been recorded.

When FEAT_SPE_EXC is implemented, this field indicates a management event for EL1.

If FEAT_SPE_EXC is implemented and the SPE Profiling exception for EL1 is enabled, then when this field is 1, an SPE Profiling exception for EL1 is pending

If FEAT_SPE_EXC is not implemented or the SPE Profiling exception for EL1 is disabled, then this field drives the **PMBIRQ** Profiling Buffer interrupt request signal.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COLL, bit [16]

Collision detected.

COLL	Meaning
0b0	No collision events detected.
0b1	At least one collision event was recorded.

The reset behavior of this field is:

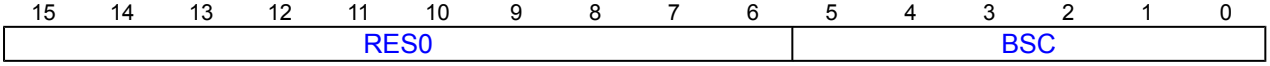
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the Profiling Buffer management event.

The syndrome contents for each Profiling Buffer management event are described in the following sections.

MSS encoding for other Profiling Buffer management events



Bits [15:6]

Reserved, RES0.

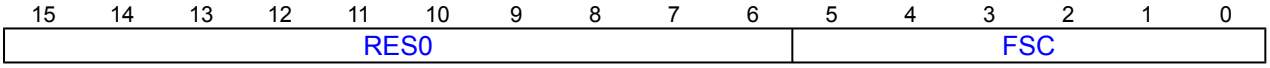
BSC, bits [5:0]

Profiling Buffer status code

BSC	Meaning
0b000000	Collection not stopped, or access not allowed.
0b000001	Profiling Buffer filled.
0b000100	Buffer size. The requested Profiling Buffer size was too large.

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer



Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented

0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented

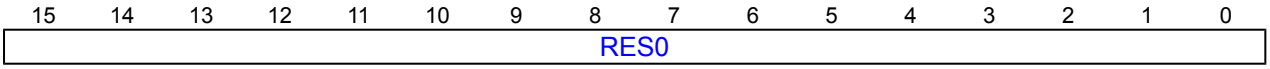
All other values are reserved.

It is IMPLEMENTATION DEFINED whether each of the Access Flag fault, asynchronous External abort and synchronous External abort, Alignment fault, and TLB Conflict abort values can be generated by the PE. For more information see 'Faults and Watchpoints'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to Profiling Buffer



Bits [15:0]

Reserved, RES0.

MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing PMBSR_EL1

When the Effective value of HCR_EL2.E2H is 1 and FEAT_SPE_EXC is implemented, without explicit synchronization, accesses from EL3 using the accessor name PMBSR_EL1 or PMBSR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (EffectivePMSCR_EL2_EE() == '00' || PMSCR_EL1.EE ==
'00' || EffectiveHCR_EL2_NVx() == '111') then
        X[t, 64] = NVMem[0x820];
    else
        X[t, 64] = PMBSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectivePMSCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        X[t, 64] = PMBSR_EL2;
    else
        X[t, 64] = PMBSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBSR_EL1;
```

```
MSR PMBSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (EffectivePMSCR_EL2_EE() == '00' || PMSCR_EL1.EE ==
'00' || EffectiveHCR_EL2_NVx() == '111') then
        NVMem[0x820] = X[t, 64];
    else
        PMBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectivePMSCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        PMBSR_EL2 = X[t, 64];
    else
        PMBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBSR_EL1 = X[t, 64];
```

When FEAT_SPE_EXC is implemented and FEAT_VHE is implemented

```
MRS <Xt>, PMBSR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x820];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMBSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = PMBSR_EL1;
    else
        UNDEFINED;
```

When FEAT_SPE_EXC is implemented and FEAT_VHE is implemented

```
MSR PMBSR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x820] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMBSR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        PMBSR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBSR_EL2, Profiling Buffer Syndrome Register (EL2)

The PMBSR_EL2 characteristics are:

Purpose

Provides syndrome information to software for a Profiling Buffer management event.

Configuration

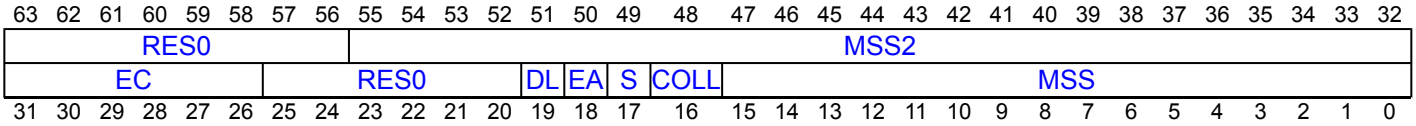
This register is present only when FEAT_SPE_EXC is implemented. Otherwise, direct accesses to PMBSR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

PMBSR_EL2 is a 64-bit register.

Field descriptions



Bits [63:56]

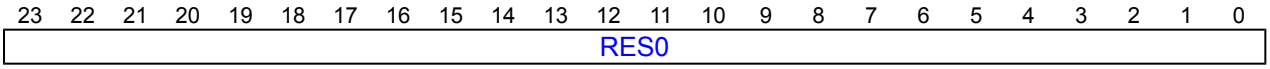
Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

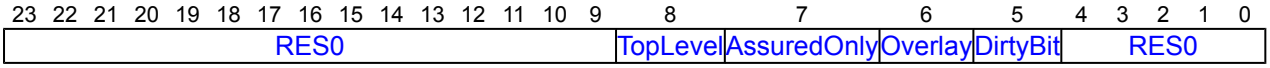
MSS2 encoding for other Profiling Buffer management events



Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer



Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, PMBSR_EL2.EC == 0b100101, and GetPMBSR_EL2_FSC() IN {0b0011xx}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetPMBSR_EL2_FSC() IN {0b0011xx}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetPMBSR_EL2_FSC() IN {0b0011xx}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

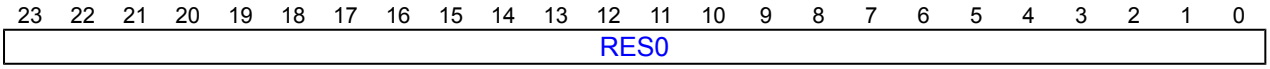
Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

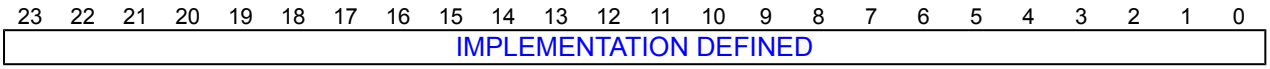
MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer



Bits [23:0]

Reserved, RES0.

MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason



IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the Profiling Buffer management event.

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other Profiling Buffer management event. All Profiling Buffer management events other than those described by the other defined Event class codes.	MSS encoding for other Profiling Buffer management events	MSS2 encoding for other Profiling Buffer management events	
0b011110	Granule Protection Check fault on write to Profiling Buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to Profiling Buffer	MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer	When FEAT_RME is implemented
0b011111	Profiling Buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	
0b100101	Stage 2 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:20]

Reserved, RES0.

DL, bit [19]

Partial record lost. Following a buffer management event other than an asynchronous External abort, indicates whether the last record written to the Profiling Buffer is complete.

DL	Meaning
0b0	PMBPTR_EL1 points to the first byte after the last complete record written to the Profiling Buffer.
0b1	Part of a record was lost because of a buffer management event or synchronous External abort. PMBPTR_EL1 might not point to the first byte after the last complete record written to the buffer, and so restarting collection might result in a data record stream that software cannot parse.

When the buffer management event was because of an asynchronous External abort, this bit is set to 1 and software must not assume that any valid data has been written to the Profiling Buffer.

This bit is RES0 if the PE never sets this bit as a result of a buffer management event caused by an asynchronous External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [18]

External abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Statistical Profiling Unit.

This bit is RES0 if the PE never sets this bit as the result of an External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [17]

Service. Indicates that a Profiling Buffer management event has been recorded.

S	Meaning
0b0	No Profiling Buffer management event for EL2 has been recorded.
0b1	A Profiling Buffer management event for EL2 has been recorded.

When FEAT_SPE_EXC is implemented, this field indicates a management event for EL2.

If the SPE Profiling exception for EL2 is enabled, then when this field is 1, an SPE Profiling exception for EL2 is pending

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COLL, bit [16]

Collision detected.

COLL	Meaning
0b0	No collision events detected.
0b1	At least one collision event was recorded.

The reset behavior of this field is:

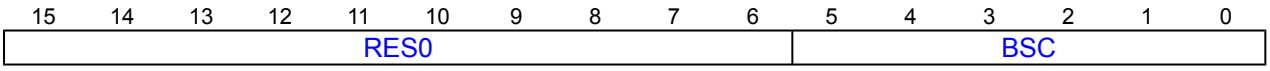
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the Profiling Buffer management event.

The syndrome contents for each Profiling Buffer management event are described in the following sections.

MSS encoding for other Profiling Buffer management events



Bits [15:6]

Reserved, RES0.

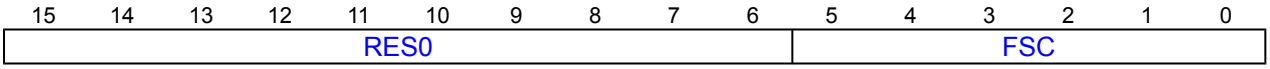
BSC, bits [5:0]

Profiling Buffer status code

BSC	Meaning
0b000000	Collection not stopped, or access not allowed.
0b000001	Profiling Buffer filled.
0b000100	Buffer size. The requested Profiling Buffer size was too large.

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer



Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented

0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented

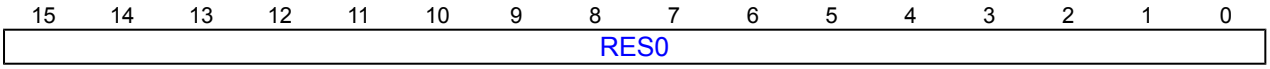
All other values are reserved.

It is IMPLEMENTATION DEFINED whether each of the Access Flag fault, asynchronous External abort and synchronous External abort, Alignment fault, and TLB Conflict abort values can be generated by the PE. For more information see 'Faults and Watchpoints'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to Profiling Buffer



Bits [15:0]

Reserved, RES0.

MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing PMBSR_EL2

When the Effective value of HCR_EL2.E2H is 1 and FEAT_SPE_EXC is implemented, without explicit synchronization, accesses from EL2 using the accessor name PMBSR_EL2 or PMBSR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBSR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE_EXC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.PMSEE == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.PMSEE == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMBSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBSR_EL2;
```

```
MSR PMBSR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE_EXC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.PMSEE == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.PMSEE == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBSR_EL2 = X[t, 64];
```

MRS <Xt>, PMBSR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (EffectivePMSCR_EL2_EE() == '00' || PMSCR_EL1.EE ==
'00' || EffectiveHCR_EL2_NVx() == '111') then
        X[t, 64] = NVMem[0x820];
    else
        X[t, 64] = PMBSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectivePMSCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        X[t, 64] = PMBSR_EL2;
    else
        X[t, 64] = PMBSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBSR_EL1;
```

MSR PMBSR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (EffectivePMSCR_EL2_EE() == '00' || PMSCR_EL1.EE ==
'00' || EffectiveHCR_EL2_NVx() == '111') then
        NVMem[0x820] = X[t, 64];
    else
        PMBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectivePMSCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        PMBSR_EL2 = X[t, 64];
    else
        PMBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBSR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBSR_EL3, Profiling Buffer Syndrome Register (EL3)

The PMBSR_EL3 characteristics are:

Purpose

Provides syndrome information to software for a Profiling Buffer management event.

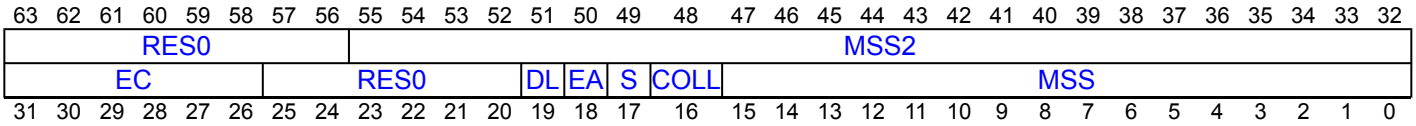
Configuration

This register is present only when FEAT_SPE_EXC is implemented and EL3 is implemented. Otherwise, direct accesses to PMBSR_EL3 are UNDEFINED.

Attributes

PMBSR_EL3 is a 64-bit register.

Field descriptions



Bits [63:56]

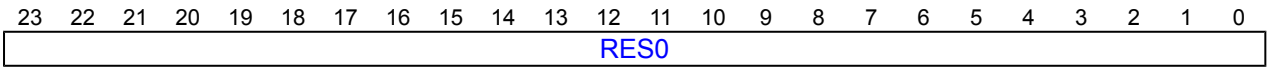
Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

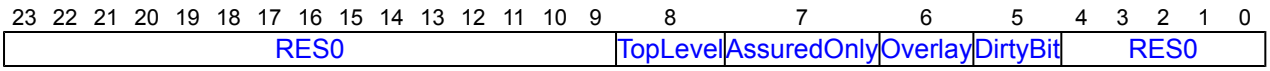
MSS2 encoding for other Profiling Buffer management events



Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer



Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, PMBSR_EL3.EC == 0b100101, and GetPMBSR_EL3_FSC() IN {0b0011xx}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetPMBSR_EL3_FSC() IN {0b0011xx}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetPMBSR_EL3_FSC() IN {0b0011xx}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

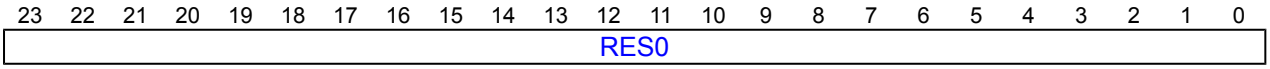
Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

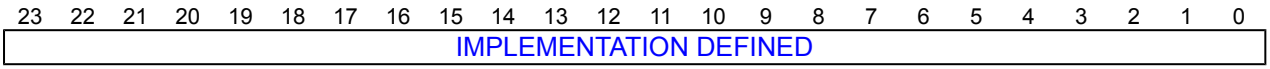
MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer



Bits [23:0]

Reserved, RES0.

MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason



IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the Profiling Buffer management event.

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other Profiling Buffer management event. All Profiling Buffer management events other than those described by the other defined Event class codes.	MSS encoding for other Profiling Buffer management events	MSS2 encoding for other Profiling Buffer management events	
0b011110	Granule Protection Check fault on write to Profiling Buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to Profiling Buffer	MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer	When FEAT_RME is implemented
0b011111	Profiling Buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	
0b100101	Stage 2 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:20]

Reserved, RES0.

DL, bit [19]

Partial record lost. Following a buffer management event other than an asynchronous External abort, indicates whether the last record written to the Profiling Buffer is complete.

DL	Meaning
0b0	PMBPTR_EL1 points to the first byte after the last complete record written to the Profiling Buffer.
0b1	Part of a record was lost because of a buffer management event or synchronous External abort. PMBPTR_EL1 might not point to the first byte after the last complete record written to the buffer, and so restarting collection might result in a data record stream that software cannot parse.

When the buffer management event was because of an asynchronous External abort, this bit is set to 1 and software must not assume that any valid data has been written to the Profiling Buffer.

This bit is RES0 if the PE never sets this bit as a result of a buffer management event caused by an asynchronous External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [18]

External abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Statistical Profiling Unit.

This bit is RES0 if the PE never sets this bit as the result of an External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [17]

Service. Indicates that a Profiling Buffer management event has been recorded.

S	Meaning
0b0	No Profiling Buffer management event for EL3 has been recorded.
0b1	A Profiling Buffer management event for EL3 has been recorded.

When FEAT_SPE_EXC is implemented, this field indicates a management event for EL3.

If the SPE Profiling exception for EL3 is enabled, then when this field is 1, an SPE Profiling exception for EL3 is pending

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COLL, bit [16]

Collision detected.

COLL	Meaning
0b0	No collision events detected.
0b1	At least one collision event was recorded.

The reset behavior of this field is:

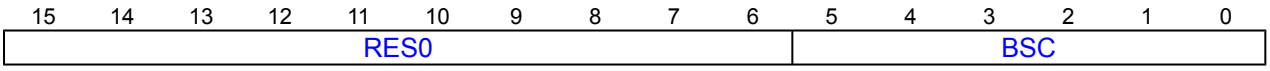
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the Profiling Buffer management event.

The syndrome contents for each Profiling Buffer management event are described in the following sections.

MSS encoding for other Profiling Buffer management events



Bits [15:6]

Reserved, RES0.

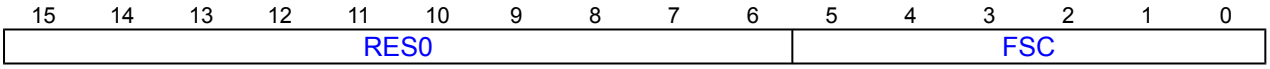
BSC, bits [5:0]

Profiling Buffer status code

BSC	Meaning
0b000000	Collection not stopped, or access not allowed.
0b000001	Profiling Buffer filled.
0b000100	Buffer size. The requested Profiling Buffer size was too large.

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer



Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented

0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented

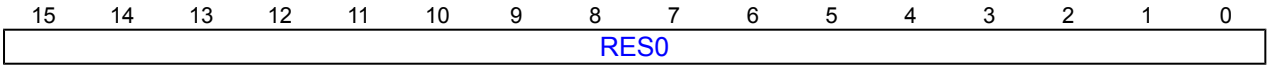
All other values are reserved.

It is IMPLEMENTATION DEFINED whether each of the Access Flag fault, asynchronous External abort and synchronous External abort, Alignment fault, and TLB Conflict abort values can be generated by the PE. For more information see 'Faults and Watchpoints'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to Profiling Buffer



Bits [15:0]

Reserved, RES0.

MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing PMBSR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBSR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1001	0b1010	0b011

```
if !(IsFeatureImplemented(FEAT_SPE_EXC) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBSR_EL3;
```

```
MSR PMBSR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1001	0b1010	0b011

```
if !(IsFeatureImplemented(FEAT_SPE_EXC) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    PMBSR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCFILTR_EL0, Performance Monitors Cycle Count Filter Register

The PMCCFILTR_EL0 characteristics are:

Purpose

Determines the modes in which the Cycle Counter, PMCCNTR_EL0, increments.

Configuration

AArch64 System register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCCFILTR[31:0].

AArch64 System register PMCCFILTR_EL0 bits [63:32] are architecturally mapped to External register PMCCFILTR_EL0[63:32] when FEAT_PMUv3_TH is implemented, or FEAT_PMUv3p8 is implemented, or FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_SME is implemented.

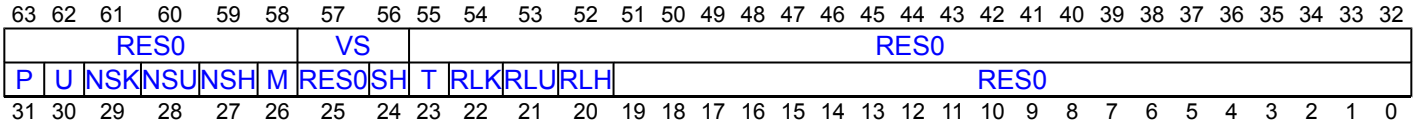
AArch64 System register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to External register PMCCFILTR_EL0[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCCFILTR_EL0 are UNDEFINED.

Attributes

PMCCFILTR_EL0 is a 64-bit register.

Field descriptions



Bits [63:58]

Reserved, RES0.

VS, bits [57:56]

When FEAT_PMUv3_SME is implemented:

SVE mode filtering. Controls counting cycles in Streaming and Non-streaming SVE modes.

VS	Meaning
0b00	This mechanism has no effect on the filtering of cycles.
0b01	The PE does not count cycles in Streaming SVE mode.
0b10	The PE does not count cycles in Non-streaming SVE mode.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [55:32]

Reserved, RES0.

P, bit [31]

EL1 filtering. Controls counting cycles in EL1.

P	Meaning
0b0	This mechanism has no effect on filtering of cycles.
0b1	The PE does not count cycles in EL1.

If Secure and Non-secure states are implemented, then counting cycles in Non-secure EL1 is further controlled by PMCCFILTR_EL0.NSK.

If FEAT_RME is implemented, then counting cycles in Realm EL1 is further controlled by PMCCFILTR_EL0.RLK.

If EL3 is implemented, then counting cycles in EL3 is further controlled by PMCCFILTR_EL0.M.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

EL0 filtering. Controls counting cycles in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of cycles.
0b1	The PE does not count cycles in EL0.

If Secure and Non-secure states are implemented, then counting cycles in Non-secure EL0 is further controlled by PMCCFILTR_EL0.NSU.

If FEAT_RME is implemented, then counting cycles in Realm EL0 is further controlled by PMCCFILTR_EL0.RLU.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting cycles in Non-secure EL1. If PMCCFILTR_EL0.NSK is not equal to PMCCFILTR_EL0.P, then the PE does not count cycles in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of cycles in Non-secure EL1.

NSK	Meaning
0b0	When PMCCFILTR_EL0.P == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.P == 1, the PE does not count cycles in Non-secure EL1.
0b1	When PMCCFILTR_EL0.P == 0, the PE does not count cycles in Non-secure EL1. When PMCCFILTR_EL0.P == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]
When EL3 is implemented:

Non-secure EL0 filtering. Controls counting cycles in Non-secure EL0. If PMCCFILTR_EL0.NSU is not equal to PMCCFILTR_EL0.U, then the PE does not count cycles in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of cycles in Non-secure EL0.

NSU	Meaning
0b0	When PMCCFILTR_EL0.U == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.U == 1, the PE does not count cycles in Non-secure EL0.
0b1	When PMCCFILTR_EL0.U == 0, the PE does not count cycles in Non-secure EL0. When PMCCFILTR_EL0.U == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]
When EL2 is implemented:

EL2 filtering. Controls counting cycles in EL2.

NSH	Meaning
0b0	The PE does not count cycles in EL2.
0b1	This mechanism has no effect on filtering of cycles.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting cycles in Secure EL2 is further controlled by PMCCFILTR_EL0.SH.

If FEAT_RME is implemented, then counting cycles in Realm EL2 is further controlled by PMCCFILTR_EL0.RLH.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]
When EL3 is implemented:

EL3 filtering. Controls counting cycles in EL3. If PMCCFILTR_EL0.M is not equal to PMCCFILTR_EL0.P, then the PE does not count cycles in EL3. Otherwise, this mechanism has no effect on filtering of cycles in EL3.

M	Meaning
0b0	When PMCCFILTR_EL0.P == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.P == 1, the PE does not count cycles in EL3.
0b1	When PMCCFILTR_EL0.P == 0, the PE does not count cycles in EL3. When PMCCFILTR_EL0.P == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [25]

Reserved, RES0.

SH, bit [24]

When EL3 is implemented and FEAT_SEL2 is implemented:

Secure EL2 filtering. Controls counting cycles in Secure EL2. If PMCCFILTR_EL0.SH is equal to PMCCFILTR_EL0.NSH, then the PE does not count cycles in Secure EL2. Otherwise, this mechanism has no effect on filtering of cycles in Secure EL2.

SH	Meaning
0b0	When PMCCFILTR_EL0.NSH == 0, the PE does not count cycles in Secure EL2. When PMCCFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of cycles.
0b1	When PMCCFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.NSH == 1, the PE does not count cycles in Secure EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When Secure EL2 is not implemented, access to this field is **RES0**.

Otherwise:

Reserved, RES0.

T, bit [23]

When FEAT_TME is implemented:

Non-Transactional state filtering field. Controls counting of cycles in Non-transactional state.

T	Meaning
0b0	This field has no effect on the filtering of cycles.
0b1	Do not count Attributable cycles in Non-transactional state.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLK, bit [22]

When FEAT_RME is implemented:

Realm EL1 filtering. Controls counting cycles in Realm EL1. If PMCCFILTR_EL0.RLK is not equal to PMCCFILTR_EL0.P, then the PE does not count cycles in Realm EL1. Otherwise, this mechanism has no effect on filtering of cycles in Realm EL1.

RLK	Meaning
0b0	When PMCCFILTR_EL0.P == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.P == 1, the PE does not count cycles in Realm EL1.
0b1	When PMCCFILTR_EL0.P == 0, the PE does not count cycles in Realm EL1. When PMCCFILTR_EL0.P == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting cycles in Realm EL0. If PMCCFILTR_EL0.RLU is not equal to PMCCFILTR_EL0.U, then the PE does not count cycles in Realm EL0. Otherwise, this mechanism has no effect on filtering of cycles in Realm EL0.

RLU	Meaning
0b0	When PMCCFILTR_EL0.U == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.U == 1, the PE does not count cycles in Realm EL0.
0b1	When PMCCFILTR_EL0.U == 0, the PE does not count cycles in Realm EL0. When PMCCFILTR_EL0.U == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]

When FEAT_RME is implemented:

Realm EL2 filtering. Controls counting cycles in Realm EL2. If PMCCFILTR_EL0.RLH is equal to PMCCFILTR_EL0.NSH, then the PE does not count cycles in Realm EL2. Otherwise, this mechanism has no effect on filtering of cycles in Realm EL2.

RLH	Meaning
0b0	When PMCCFILTR_EL0.NSH == 0, the PE does not count cycles in Realm EL2. When PMCCFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of cycles.
0b1	When PMCCFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.NSH == 1, the PE does not count cycles in Realm EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:0]

Reserved, RES0.

Accessing PMCCFILTR_EL0

PMCCFILTR_EL0 can also be accessed by using PMXEVTYPER_EL0 with PMSELR_EL0.SEL set to 0b11111.

Permitted reads and writes of PMCCFILTR_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.C == 0.

Permitted writes of PMCCFILTR_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.{UEN,CR} == {1,1}.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCCFILTR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b1111	0b111

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.C == '0'
then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMCCFILTR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCCFILTR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCCFILTR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCCFILTR_EL0;
```

```
MSR PMCCFILTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b1111	0b111

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.C == '0'
|| PMUSERENR_EL0.CR == '1') then
        return;
    else
        PMCCFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMCCFILTR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCNTR_EL0, Performance Monitors Cycle Count Register

The PMCCNTR_EL0 characteristics are:

Purpose

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' for more information.

PMCCFILTR_EL0 determines the modes and states in which the PMCCNTR_EL0 can increment.

Configuration

AArch64 System register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch32 System register PMCCNTR[63:0].

AArch64 System register PMCCNTR_EL0 bits [63:0] are architecturally mapped to External register PMCCNTR_EL0[63:0].

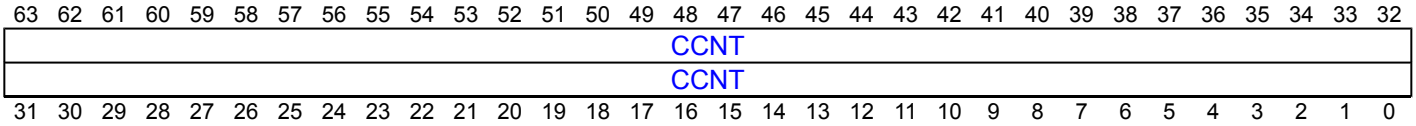
This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCCNTR_EL0 are UNDEFINED.

All counters are subject to any changes in clock frequency, including clock stopping caused by the WFI and WFE instructions. This means that it is CONSTRAINED UNPREDICTABLE whether or not PMCCNTR_EL0 continues to increment when clocks are stopped by WFI and WFE instructions.

Attributes

PMCCNTR_EL0 is a 64-bit register.

Field descriptions



CCNT, bits [63:0]

Cycle count. Depending on the values of PMCR_EL0.{LC,D}, this field increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to PMCR_EL0.C sets this field to 0.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMCCNTR_EL0

Permitted reads and writes of PMCCNTR_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.C == 0.

Permitted writes of PMCCNTR_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.{UEN,CR} == {1,1}.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCCNTR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,CR,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<CR,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.C == '0'
then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMCCNTR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCCNTR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCCNTR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCCNTR_EL0;
```

```
MSR PMCCNTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.C == '0'
|| PMUSERENR_EL0.CR == '1') then
        return;
    else
        PMCCNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMCCNTR_EL0 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCNTSVR_EL1, Performance Monitors Cycle Count Saved Value Register

The PMCCNTSVR_EL1 characteristics are:

Purpose

Captures the PMU Cycle counter, PMCCNTR_EL0.

Configuration

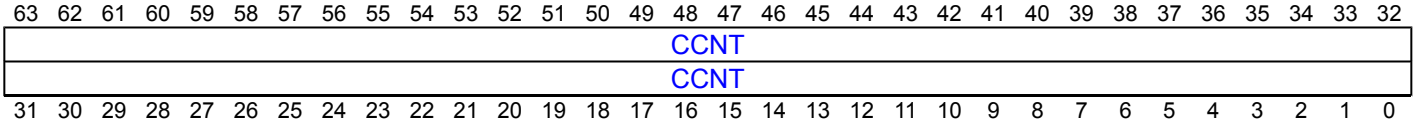
AArch64 System register PMCCNTSVR_EL1 bits [63:0] are architecturally mapped to External register PMCCNTSVR_EL1[63:0].

This register is present only when FEAT_PMUv3_SS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCCNTSVR_EL1 are UNDEFINED.

Attributes

PMCCNTSVR_EL1 is a 64-bit register.

Field descriptions



CCNT, bits [63:0]

Sampled Cycle Count. The value of PMCCNTR_EL0 at the last successful Capture event.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMCCNTSVR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCCNTSVR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1110	0b1011	0b111

```
if !(IsFeatureImplemented(FEAT_PMUv3_SS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMSSDATA == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCCNTSVR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCCNTSVR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCCNTSVR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID0_EL0, Performance Monitors Common Event Identification Register 0

The PMCEID0_EL0 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the ranges `0x0000` to `0x001F` and `0x4000` to `0x401F`.

For more information about the Common events and the use of the PMCEID<n>_EL0 registers see 'The PMU event number space and common events'.

Configuration

AArch64 System register PMCEID0_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCEID0[31:0].

AArch64 System register PMCEID0_EL0 bits [63:32] are architecturally mapped to AArch32 System register PMCEID2[31:0].

AArch64 System register PMCEID0_EL0 bits [31:0] are architecturally mapped to External register PMCEID0[31:0].

AArch64 System register PMCEID0_EL0 bits [63:32] are architecturally mapped to External register PMCEID2[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCEID0_EL0 are UNDEFINED.

Attributes

PMCEID0_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	
IDhi31	IDhi30	IDhi29	IDhi28	IDhi27	IDhi26	IDhi25	IDhi24	IDhi23	IDhi22	IDhi21	IDhi20	IDhi19	IDhi18	IDhi17	IDhi16	IDhi15	IDhi14	IDhi13	IDh
ID31	ID30	ID29	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13	ID
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	

IDhi<n>, bit [n+32], for n = 31 to 0

When FEAT_PMUv3p1 is implemented:

IDhi[n] corresponds to Common event (`0x4000` + n).

For each bit:

IDhi<n>	Meaning
`0b0`	The Common event is not implemented, or not counted.
`0b1`	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Otherwise:

Reserved, RES0.

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event n.

For each bit:

ID<n>	Meaning
0b0	The Common event is not implemented, or not counted.
0b1	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Accessing PMCEID0_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCEID0_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.TID == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID0_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID0_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID0_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCEID0_EL0;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID1_EL0, Performance Monitors Common Event Identification Register 1

The PMCEID1_EL0 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the ranges `0x0020` to `0x003F` and `0x4020` to `0x403F`.

For more information about the Common events and the use of the PMCEID<n>_EL0 registers see 'The PMU event number space and common events'.

Configuration

AArch64 System register PMCEID1_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCEID1[31:0].

AArch64 System register PMCEID1_EL0 bits [63:32] are architecturally mapped to AArch32 System register PMCEID3[31:0].

AArch64 System register PMCEID1_EL0 bits [31:0] are architecturally mapped to External register PMCEID1[31:0].

AArch64 System register PMCEID1_EL0 bits [63:32] are architecturally mapped to External register PMCEID3[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCEID1_EL0 are UNDEFINED.

Attributes

PMCEID1_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	
IDhi31	IDhi30	IDhi29	IDhi28	IDhi27	IDhi26	IDhi25	IDhi24	IDhi23	IDhi22	IDhi21	IDhi20	IDhi19	IDhi18	IDhi17	IDhi16	IDhi15	IDhi14	IDhi13	ID
ID31	ID30	ID29	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13	I
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	

IDhi<n>, bit [n+32], for n = 31 to 0

When FEAT_PMUv3p1 is implemented:

IDhi[n] corresponds to Common event (`0x4020` + n).

For each bit:

IDhi<n>	Meaning
`0b0`	The Common event is not implemented, or not counted.
`0b1`	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Otherwise:

Reserved, RES0.

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event (0x0020 + n).

For each bit:

ID<n>	Meaning
0b0	The Common event is not implemented, or not counted.
0b1	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Accessing PMCEID1_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCEID1_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.TID == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID1_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID1_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID1_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCEID1_EL0;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTENCLR_EL0, Performance Monitors Count Enable Clear Register

The PMCNTENCLR_EL0 characteristics are:

Purpose

Allows software to disable the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which counters are enabled.

Configuration

AArch64 System register PMCNTENCLR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[63:0].

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENCLR[31:0].

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENSET[31:0].

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to External register PMCNTENCLR_EL0[31:0].

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to External register PMCNTENSET_EL0[31:0].

AArch64 System register PMCNTENCLR_EL0 bits [63:32] are architecturally mapped to External register PMCNTENCLR_EL0[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

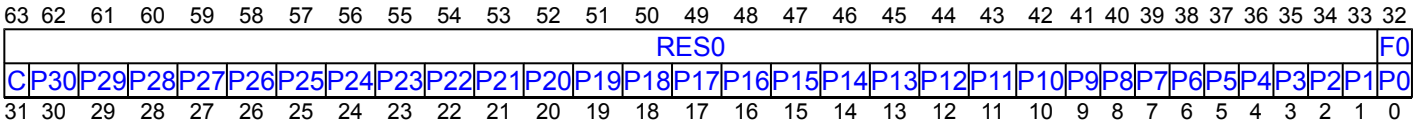
AArch64 System register PMCNTENCLR_EL0 bits [63:32] are architecturally mapped to External register PMCNTENSET_EL0[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCNTENCLR_EL0 are UNDEFINED.

Attributes

PMCNTENCLR_EL0 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

PMICNTR_EL0 disable. On writes, allows software to disable PMICNTR_EL0. On reads, returns the PMICNTR_EL0 enable status.

F0	Meaning
0b0	PMICNTR_EL0 disabled.
0b1	PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - PSTATE.EL != EL3.
  - MDCR_EL3.EnPM2 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 0 or PMUACR_EL1.F0 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or (HDFGRTR2_EL2.nPMICFILTR_EL0 == 0 and HDFGWTR2_EL2.nPMICFILTR_EL0 == 0).
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL == EL0.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGRTR2_EL2.nPMICFILTR_EL0 == 0.
  - PMUSERENR_EL0.IR == 1.
- Access to this field is **WO/RAZ** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGRTR2_EL2.nPMICFILTR_EL0 == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGWTR2_EL2.nPMICFILTR_EL0 == 0.
- Access to this field is **RO** if all the following are true:
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.IR == 1.
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 disable. On writes, allows software to disable PMCCNTR_EL0. On reads, returns the PMCCNTR_EL0 enable status.

C	Meaning
0b0	PMCCNTR_EL0 disabled.
0b1	PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.C == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,CR] == 0b11.
- Otherwise, access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

PMEVCNTR<m>_EL0 disable. On writes, allows software to disable PMEVCNTR<m>_EL0. On reads, returns the PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 disabled.
0b1	PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.P<m> == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,ER] == 0b11.
- Otherwise, access to this field is **W1C**.

Accessing PMCNTENCLR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCNTENCLR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCNTENCLR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCNTENCLR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCNTENCLR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCNTENCLR_EL0;
```

```
MSR PMCNTENCLR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMCNTENCLR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTENSET_EL0, Performance Monitors Count Enable Set Register

The PMCNTENSET_EL0 characteristics are:

Purpose

Allows software to enable the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which counters are enabled.

Configuration

AArch64 System register PMCNTENSET_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[63:0].

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENSET[31:0].

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENCLR[31:0].

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to External register PMCNTENSET_EL0[31:0].

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to External register PMCNTENCLR_EL0[31:0].

AArch64 System register PMCNTENSET_EL0 bits [63:32] are architecturally mapped to External register PMCNTENSET_EL0[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

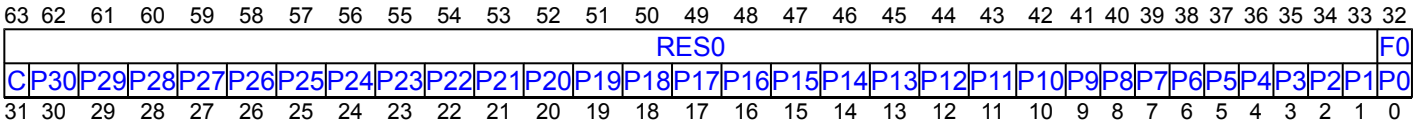
AArch64 System register PMCNTENSET_EL0 bits [63:32] are architecturally mapped to External register PMCNTENCLR_EL0[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCNTENSET_EL0 are UNDEFINED.

Attributes

PMCNTENSET_EL0 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

PMICNTR_EL0 enable. On writes, allows software to enable PMICNTR_EL0. On reads, returns the PMICNTR_EL0 enable status.

F0	Meaning
0b0	PMICNTR_EL0 disabled.
0b1	PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - PSTATE.EL != EL3.
  - MDCR_EL3.EnPM2 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 0 or PMUACR_EL1.F0 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or (HDFGRTR2_EL2.nPMICFILTR_EL0 == 0 and HDFGWTR2_EL2.nPMICFILTR_EL0 == 0).
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL == EL0.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGRTR2_EL2.nPMICFILTR_EL0 == 0.
  - PMUSERENR_EL0.IR == 1.
- Access to this field is **WO/RAZ** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGRTR2_EL2.nPMICFILTR_EL0 == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGWTR2_EL2.nPMICFILTR_EL0 == 0.
- Access to this field is **RO** if all the following are true:
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.IR == 1.
- Otherwise, access to this field is **W1S**.

Otherwise:

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 enable. On writes, allows software to enable PMCCNTR_EL0. On reads, returns the PMCCNTR_EL0 enable status.

C	Meaning
0b0	PMCCNTR_EL0 disabled.
0b1	PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.C == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,CR] == 0b11.
- Otherwise, access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

PMEVCNTR<m>_EL0 enable. On writes, allows software to enable PMEVCNTR<m>_EL0. On reads, returns the PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 disabled.
0b1	PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.P<m> == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,ER] == 0b11.
- Otherwise, access to this field is **W1S**.

Accessing PMCNTENSET_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCNTENSET_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCNTENSET_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCNTENSET_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCNTENSET_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCNTENSET_EL0;
```

```
MSR PMCNTENSET_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMCNTENSET_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCR_EL0, Performance Monitors Control Register

The PMCR_EL0 characteristics are:

Purpose

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Configuration

AArch64 System register PMCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCR[31:0].

AArch64 System register PMCR_EL0 bits [31:0] are architecturally mapped to External register PMCR_EL0[31:0].

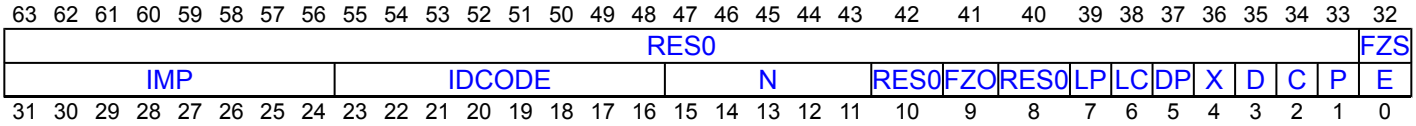
AArch64 System register PMCR_EL0 bits [63:32] are architecturally mapped to External register PMCR_EL0[63:32] when FEAT_PMUv3_EXT64 is implemented.

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCR_EL0 are UNDEFINED.

Attributes

PMCR_EL0 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

FZS, bit [32]

When FEAT_SPEv1p2 is implemented:

Freeze-on-SPE event. Stop counters when PMBLIMITR_EL1.{PMFZ,E} is {1,1} and profiling is stopped.

FZS	Meaning
0b0	Do not freeze on a Statistical Profiling Buffer Management event.
0b1	Affected counters do not count following a Statistical Profiling Buffer Management event.

The pseudocode function `SPEProfilingStopped` describes when profiling is stopped.

The counters affected by this field are:

- The event counters in the first range.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- If FEAT_SPE_DPFZS is implemented and PMCR_EL0.DP is 1, the cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

When FEAT_SPE_DPFZS is not implemented or PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA32 is implemented, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IMP, bits [31:24]
When FEAT_PMUv3p7 is not implemented:

Implementer code.

If this field is zero, then PMCR_EL0.IDCODE is RES0 and software must use MIDR_EL1 to identify the PE.

Otherwise, this field and PMCR_EL0.IDCODE identify the PMU implementation to software. The implementer codes are allocated by Arm. A nonzero value has the same interpretation as MIDR_EL1.Implementer.

Arm deprecates use of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

IDCODE, bits [23:16]
When PMCR_EL0.IMP != 0b00000000:

Identification code. Arm deprecates use of this field.

Each implementer must maintain a list of identification codes that are specific to the implementer. A specific implementation is identified by the combination of the implementer code and the identification code.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

N, bits [15:11]

Indicates the number of event counters implemented. This value is in the range of `0b00000-0b11111`. If the value is `0b00000`, then only PMCCNTR_EL0 is implemented. If the value is `0b11111`, then PMCCNTR_EL0 and 31 event counters are implemented.

When EL2 is implemented and enabled for the current Security state, reads of this field from EL1 and EL0 return the Effective value of MDCR_EL2.HPMN.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [10]

Reserved, RES0.

FZO, bit [9]

When FEAT_PMUv3p7 is implemented:

Freeze-on-overflow. Stop event counters on overflow.

FZO	Meaning
0b0	Do not freeze on overflow.
0b1	Affected counters do not count when any of the following applies: • For any event counter PMEVCNTR<m>_EL0 in the first range, PMOVSCLR_EL0[m] is 1, and either FEAT_SEBEP is not implemented or PMEVTYPER<m>_EL0.SYNC is 0. • FEAT_PMUv3_ICNTR is implemented, PMOVSCLR_EL0.F0 is 1, and either FEAT_SEBEP is not implemented or PMICFILTR_EL0.SYNC is 0.

The counters affected by this field are:

- The event counters in the first range.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- If PMCR_EL0.DP is 1, the cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

LP, bit [7]

When FEAT_PMUv3p5 is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by PMOVSCLR_EL0.P[n].

LP	Meaning
0b0	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[31:0].
0b1	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[63:0].

When FEAT_EBEP is implemented and the PMU Profiling exception is enabled, the Effective value of this field is 1.

The counters affected by this field are the event counters in the first range. For more information about event counter ranges, see MDCR_EL2.HPMN.

Other event counters and PMCCNTR_EL0 are not affected by this field.

When FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0 is not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LC, bit [6]
When FEAT_AA32 is implemented:

Long cycle counter enable. Determines when unsigned overflow is recorded by PMOVSCLR_EL0.C.

LC	Meaning
0b0	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[31:0].
0b1	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[63:0].

When FEAT_EBEP is implemented and the PMU Profiling exception is enabled, the Effective value of this field is 1.

Arm deprecates use of PMCR_EL0.LC = 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

DP, bit [5]
When EL3 is implemented or (FEAT_PMUv3p1 is implemented and EL2 is implemented):

Disable cycle counter when event counting is prohibited.

DP	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is disabled in prohibited regions and when event counting is frozen: • If FEAT_PMUv3p1 is implemented, EL2 is implemented, and MDCR_EL2.HPMD is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL2. • If FEAT_SPE_DPFZS is implemented and event counting is frozen by PMCR_EL0.FZS, then cycle counting by PMCCNTR_EL0 is disabled. • If FEAT_PMUv3p7 is implemented and event counting is frozen by PMCR_EL0.FZO, then cycle counting by PMCCNTR_EL0 is disabled. • If FEAT_PMUv3p7 is implemented, EL3 is implemented, and MDCR_EL3.MPMX is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL3. • If EL3 is implemented, MDCR_EL3.SPME is 0, and either FEAT_PMUv3p7 is not implemented or MDCR_EL3.MPMX is 0, then cycle counting by PMCCNTR_EL0 is disabled at EL3 and in Secure state.

The conditions when this field disables the cycle counter are the same as when event counting by an event counter in the first range is prohibited or frozen. For more information about event counter ranges, see MDCR_EL2.HPMN.

If FEAT_PMUv3p7 and FEAT_SPEv1p2 are implemented, meaning PMCR_EL0.FZS is implemented, and FEAT_SPE_DPFZS is not implemented, then cycle counting by PMCCNTR_EL0 is not affected by PMCR_EL0.FZS.

For more information, see 'Prohibiting event and cycle counting'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

X, bit [4]

When the implementation includes a PMU event export bus:

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.

X	Meaning
0b0	Do not export events.
0b1	Export events where not prohibited.

This field enables the exporting of events over an IMPLEMENTATION DEFINED PMU event export bus to another device.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

If FEAT_ETE is implemented, this field does not affect the use of PMU events as an External Input by the trace unit.

If FEAT_ETMv4 is implemented, this field does affect the use of PMU events as an External Input by the trace unit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

D, bit [3]

When FEAT_AA32 is implemented:

Clock divider.

D	Meaning
0b0	When enabled, PMCCNTR_EL0 counts every clock cycle.
0b1	When enabled, PMCCNTR_EL0 counts once every 64 clock cycles.

If the Effective value of PMCR_EL0.LC is 1, then this field is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR_EL0.D = 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

C, bit [2]

Cycle counter reset. The effects of writing to this field are:

C	Meaning
0b0	No action.
0b1	Reset PMCCNTR_EL0 to zero.

Note

Resetting PMCCNTR_EL0 does not change the cycle counter overflow field. The value of PMCR_EL0.LC is ignored, and bits [63:0] of the cycle counter are reset.

Access to this field is **WO/RAZ**.

P, bit [1]

Event counter reset.

P	Meaning
0b0	No action.
0b1	Reset all affected event counters PMEVCNTR<n>_EL0 to zero.

The event counters affected by this field are:

- All event counters in the first range.
- If any of the following are true, all event counters in the second range:
  - EL2 is disabled or not implemented in the current Security state.
  - The PE is executing at EL2 or EL3.

Writes to this field do not affect other event counters, the cycle counter PMCCNTR_EL0, or the instruction counter PMICNTR_EL0.

For more information about event counter ranges, see MDCR_EL2.HPMN.

Note

Resetting the event counters does not change the event counter overflow fields. If FEAT_PMUv3p5 is implemented, the values of MDCR_EL2.HLP and PMCR_EL0.LP are ignored, and bits [63:0] of all affected event counters are reset.

Access to this field is **WO/RAZ**.

E, bit [0]

Enable.

E	Meaning
0b0	Affected counters are disabled and do not count.
0b1	Affected counters are enabled by PMCNTENSET_EL0.

The counters affected by this field are:

- The event counters in the first range. For more information about event counter ranges, see MDCR_EL2.HPMN.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- The cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing PMCR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' || (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN ==
'1') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCR_EL0;
```

```
MSR PMCR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' || (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN ==
'1') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMCR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMCR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMECR_EL1, Performance Monitors Extended Control Register (EL1)

The PMECR_EL1 characteristics are:

Purpose

Provides EL1 configuration options for the Performance Monitors.

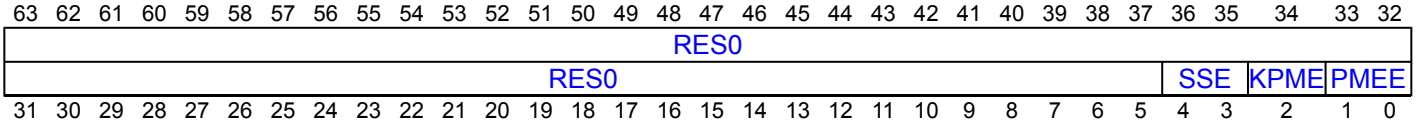
Configuration

This register is present only when (FEAT_EBEP is implemented or FEAT_PMUv3_SS is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to PMECR_EL1 are UNDEFINED.

Attributes

PMECR_EL1 is a 64-bit register.

Field descriptions



Bits [63:5]

Reserved, RES0.

SSE, bits [4:3]

When FEAT_PMUv3_SS is implemented:

Snapshot Enable. Controls the generation of Capture events.

SSE	Meaning
0b00	Capture events are disabled.
0b10	Capture events are enabled and prohibited.
0b11	Capture events are enabled and allowed.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset:
  - When the highest implemented Exception level is EL1, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

KPME, bit [2]

When FEAT_EBEP is implemented:

Local (Kernel) PMU Exception Enable. Enables PMU Profiling exceptions taken to the current Exception level.

KPME	Meaning
0b0	PMU Profiling exceptions taken to the current Exception level are disabled.
0b1	PMU Profiling exceptions taken to the current Exception level are not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMEE, bits [1:0]

When FEAT_EBEP is implemented:

Performance Monitors Exception Enable. Controls the generation of the **PMUIRQ** signal and the PMU Profiling exception at EL0 and EL1.

PMEE	Meaning
0b00	The **PMUIRQ** signal is asserted on a PMU overflow, and the PMU Profiling exception is disabled.
0b10	The **PMUIRQ** signal is deasserted, and the PMU Profiling exception is disabled.
0b11	The **PMUIRQ** signal is deasserted, and the PMU Profiling exception is enabled.

All other values are reserved.

This field is ignored by the PE when any of the following are true:

- All of the following are true:
  - EL3 is implemented.
  - MDCR_EL3.PMEE != 0b01.
- All of the following are true:
  - EL2 is implemented and enabled in the current Security State.
  - MDCR_EL2.PMEE != 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing PMECR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMECR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b101

```
if !((IsFeatureImplemented(FEAT_EBEP) || IsFeatureImplemented(FEAT_PMUv3_SS)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMECR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMECR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMECR_EL1;
```

```
MSR PMECR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b101

```
if !((IsFeatureImplemented(FEAT_EBEP) || IsFeatureImplemented(FEAT_PMUv3_SS)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nPMECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMECR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVCNTR<n>_EL0, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n>_EL0 characteristics are:

Purpose

Holds event counter n, which counts events, where n is 0 to 30.

Configuration

AArch64 System register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVCNTR<n>[31:0].

AArch64 System register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to External register PMEVCNTR<n>_EL0[31:0].

AArch64 System register PMEVCNTR<n>_EL0 bits [63:32] are architecturally mapped to External register PMEVCNTR<n>_EL0[63:32] when FEAT_PMUv3p5 is implemented.

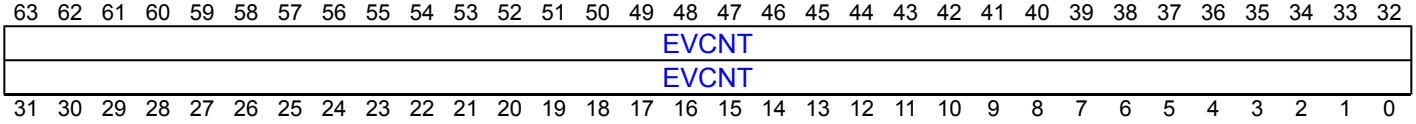
This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMEVCNTR<n>_EL0 are UNDEFINED.

Attributes

PMEVCNTR<n>_EL0 is a 64-bit register.

Field descriptions

When FEAT_PMUv3p5 is implemented:



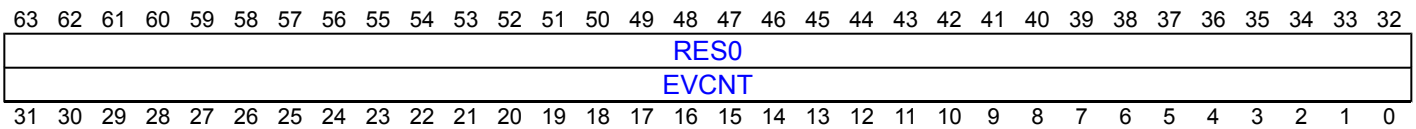
EVCNT, bits [63:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:



Bits [63:32]

Reserved, RES0.

EVCNT, bits [31:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMEVCNTR<n>_EL0

PMEVCNTR<n>_EL0 can also be accessed by using PMXEVCNTR_EL0 with PMSELR_EL0.SEL set to the value of <n>.

If FEAT_FGT is implemented and <n> is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMEVCNTR<n>_EL0 is as follows:

- If <n> is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and <n> is greater than or equal to the number of accessible event counters, then reads and writes of PMEVCNTR<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if <n> is an UNKNOWN value less-than-or-equal-to the index of the highest accessible event counter.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMEVCNTR<n>_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.P<n> == 0.

Permitted writes of PMEVCNTR<n>_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.{UEN,ER} == {1,1}.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.{UEN,ER,EN}.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible event counters. Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see MDCR_EL2.HPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMEVCNTR<m>_EL0 ; Where m = 0-30
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b10:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1[m] == '0'
then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMEVCNTR_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTR_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTR_EL0[m];
elsif PSTATE.EL == EL3 then
```

```
    X[t, 64] = PMEVCNTR_EL0[m];
```

```
MSR PMEVCNTR<m>_EL0, <Xt> ; Where m = 0-30
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b10:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1[m] == '0'
|| PMUSERENR_EL0.ER == '1') then
        return;
    else
        PMEVCNTR_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVCNTR_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVCNTR_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
```

```
PMEVCNTR_EL0[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVCNTSVR<n>_EL1, Performance Monitors Event Count Saved Value Registers, n = 0 - 30

The PMEVCNTSVR<n>_EL1 characteristics are:

Purpose

Captures the PMU Event counter <n>, PMEVCNTR<n>_EL0.

Configuration

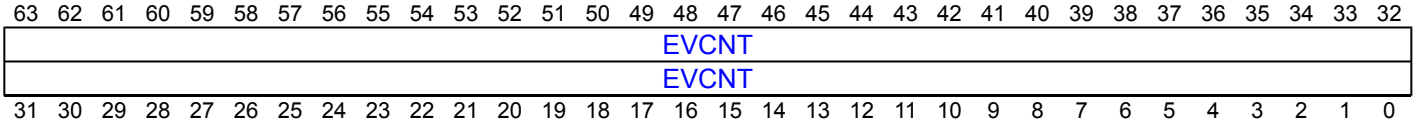
AArch64 System register PMEVCNTSVR<n>_EL1 bits [63:0] are architecturally mapped to External register PMEVCNTSVR<n>_EL1[63:0].

This register is present only when FEAT_PMUv3_SS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMEVCNTSVR<n>_EL1 are UNDEFINED.

Attributes

PMEVCNTSVR<n>_EL1 is a 64-bit register.

Field descriptions



EVCNT, bits [63:0]

Sampled Event Count. The value of PMEVCNTR<n>_EL0 at the last successful Capture event.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMEVCNTSVR<n>_EL1

If <n> is greater-than-or-equal-to the Effective value of PMCCR.EPMN, then direct reads of PMEVCNTSVR<n>_EL1 are UNDEFINED.

Otherwise, direct reads of PMEVCNTSVR<n>_EL1 generate a Trap exception to EL2 when all of the following are true:

- <n> is greater-than-or-equal-to the number of snapshot registers accessible at the current Exception level.
- EL2 is implemented and enabled in the current Security state.
- The access is from EL1.

Note

If EL2 is implemented and enabled in the current Security state, MDCR_EL2.HPMN identifies the number of accessible snapshot registers at EL1. Otherwise, the number of accessible snapshot registers is the number of implemented event counters. See MDCR_EL2.HPMN for more details.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMEVCNTSVR<m>_EL1 ; Where m = 0-30
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1110	0b10:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_PMUv3_SS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMSSDATA == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTSVR_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTSVR_EL1[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMEVCNTSVR_EL1[m];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30

The PMEVTYPER<n>_EL0 characteristics are:

Purpose

Configures event counter n, where n is 0 to 30.

Configuration

AArch64 System register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVTYPER<n>[31:0].

AArch64 System register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to External register PMEVTYPER<n>_EL0[31:0].

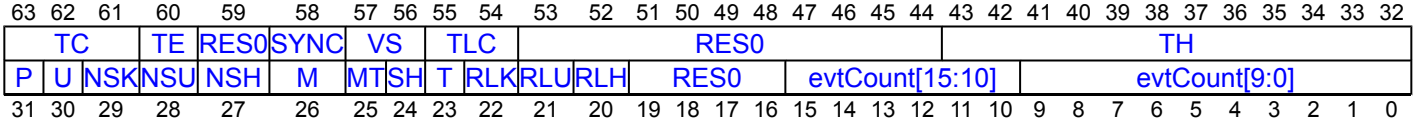
AArch64 System register PMEVTYPER<n>_EL0 bits [63:32] are architecturally mapped to External register PMEVTYPER<n>_EL0[63:32] when FEAT_PMUv3_TH is implemented, or FEAT_PMUv3p8 is implemented, or FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_SME is implemented.

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMEVTYPER<n>_EL0 are UNDEFINED.

Attributes

PMEVTYPER<n>_EL0 is a 64-bit register.

Field descriptions



TC, bits [63:61]

When FEAT_PMUv3_TH is implemented, (FEAT_PMUv3_EDGE is not implemented or PMEVTYPER<n>_EL0.TE == 0), and (FEAT_PMUv3_TH2 is not implemented, or n is even, or PMEVTYPER<n>_EL0.TLC IN {0b0x}):

Threshold Control. Defines the threshold function. In the description of this field:

- V_B[n] is the value the event specified by PMEVTYPER<n>_EL0 would increment event counter n by on a processor cycle if the threshold function is disabled.
- For odd values of n, V[n-1] is the value that event counter n-1 increments by on the same processor cycle. V[n-1] is the result of applying the threshold and edge functions on event counter n-1. If event counter n-1 is disabled then V[n-1] is zero. V[n-1] is not defined for even values of n.
- TH[n] is the value of PMEVTYPER<n>_EL0.TH.

TC	Meaning
0b000	Not-equal. The counter increments by V_B[n] on each processor cycle when V_B[n] is not equal to TH[n].
0b001	Not-equal, count. The counter increments by 1 on each processor cycle when V_B[n] is not equal to TH[n].
0b010	Equals. The counter increments by V_B[n] on each processor cycle when V_B[n] is equal to TH[n].
0b011	Equals, count. The counter increments by 1 on each processor cycle when V_B[n] is equal to TH[n].
0b100	Greater-than-or-equal. The counter increments by V_B[n] on each processor cycle when V_B[n] is greater than or equal to TH[n].
0b101	Greater-than-or-equal, count. The counter increments by 1 on each processor cycle when V_B[n] is greater than or equal to TH[n].
0b110	Less-than. The counter increments by V_B[n] on each processor cycle when V_B[n] is less than TH[n].
0b111	Less-than, count. The counter increments by 1 on each processor cycle when V_B[n] is less than TH[n].

Comparisons treat V_B[n] and TH[n] as unsigned integer values.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC[2:1] is true:

- If PMEVTYPER<n>_EL0.TC[0] is 0, then the counter increments by V_B[n].
- If PMEVTYPER<n>_EL0.TC[0] is 1, then the counter increments by 1.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC[2:1] is false:

- If FEAT_PMUv3_TH2 is implemented, n is odd, and PMEVTYPER<n>_EL0.TLC is 0b01, then the counter increments by V[n-1].
- Otherwise, the counter does not increment.

If PMEVTYPER<n>_EL0.{TC, TLC, TH} are zero then the threshold function is disabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to '000'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When FEAT_PMUv3_TH2 is implemented, PMEVTYPER<n>_EL0.TE == 0, n is odd, and PMEVTYPER<n>_EL0.TLC == 0b10:

Threshold Control. Defines the threshold function. In the description of this field:

- V_B[n] is the value the event specified by PMEVTYPER<n>_EL0 would increment event counter n by on a processor cycle if the threshold function is disabled.
- V[n-1] is the value that event counter n-1 increments by on the same processor cycle. V[n-1] is the result of applying the threshold and edge functions on event counter n-1. If event counter n-1 is disabled then V[n-1] is zero.
- TH[n] is the value of PMEVTYPER<n>_EL0.TH.

TC	Meaning
0b000	Not-equal. The counter increments by V[n-1] on each processor cycle when V_B[n] is not equal to TH[n].
0b010	Equals. The counter increments by V[n-1] on each processor cycle when V_B[n] is equal to TH[n].
0b100	Greater-than-or-equal. The counter increments by V[n-1] on each processor cycle when V_B[n] is greater than or equal to TH[n].
0b110	Less-than. The counter increments by V[n-1] on each processor cycle when V_B[n] is less than TH[n].

All other values are reserved.

Comparisons treat V_B[n] and TH[n] as unsigned integer values.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is true, the counter increments by V[n-1].

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is false, the counter does not increment.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to '000'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When FEAT_PMUv3_EDGE is implemented and PMEVTYPER<n>_EL0.TE == 1:

Threshold Control. Defines the threshold function. In the description of this field:

- V_B[n] is the value the event specified by PMEVTYPER<n>_EL0 would increment event counter n by on a processor cycle if the threshold function is disabled.
- For odd values of n, V[n-1] is the value that event counter n-1 increments by on the same processor cycle. V[n-1] is the result of applying the threshold and edge functions on event counter n-1. If event counter n-1 is disabled then V[n-1] is zero. V[n-1] is not defined for even values of n.
- TH[n] is the value of PMEVTYPER<n>_EL0.TH.

TC	Meaning
0b001	Equal to not-equal. The counter increments on each processor cycle when V_B[n] is not equal to TH[n] and V_B[n] was equal to TH[n] on the previous processor cycle.
0b010	Equal to/from not-equal. The counter increments on each processor cycle when either: • V_B[n] is not equal to TH[n] and V_B[n] was equal to TH[n] on the previous processor cycle. • V_B[n] is equal to TH[n] and V_B[n] was not equal to TH[n] on the previous processor cycle.
0b011	Not-equal to equal. The counter increments on each processor cycle when V_B[n] is equal to TH[n] and V_B[n] was not equal to TH[n] on the previous processor cycle.
0b101	Less-than to greater-than-or-equal. The counter increments on each processor cycle when V_B[n] is greater than or equal to TH[n] and V_B[n] was less than TH[n] on the previous processor cycle.
0b110	Less-than to/from greater-than-or-equal. The counter increments on each processor cycle when either: • V_B[n] is greater than or equal to TH[n] and V_B[n] was less than TH[n] on the previous processor cycle. • V_B[n] is less than TH[n] and V_B[n] was greater than or equal to TH[n] on the previous processor cycle.
0b111	Greater-than-or-equal to less-than. The counter increments on each processor cycle when V_B[n] is less than TH[n] and V_B[n] was greater than or equal to TH[n] on the previous processor cycle.

All other values are reserved.

Comparisons treat V_B[n] and TH[n] as unsigned integer values.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is true:

- If FEAT_PMUv3_TH2 is implemented, n is odd, and PMEVTYPER<n>_EL0.TLC is 0b10, then the counter increments by V[n-1].
- Otherwise, the counter increments by 1.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is false, the counter does not increment.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to '000'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TE, bit [60]

When FEAT_PMUv3_EDGE is implemented:

Threshold Edge. Enables the edge condition. When PMEVTYPER<n>_EL0.TE is 1, the event counter increments on cycles when the result of the threshold condition changes. See PMEVTYPER<n>_EL0.TC for more information.

TE	Meaning
0b0	Threshold edge condition disabled.
0b1	Threshold edge condition enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to '0'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [59]

Reserved, RES0.

SYNC, bit [58]

When FEAT_SEBEP is implemented:

Synchronous mode. Controls whether a PMU Profiling exception generated by the counter is synchronous or asynchronous.

SYNC	Meaning
0b0	Asynchronous PMU Profiling exception is enabled.
0b1	Synchronous PMU Profiling exception is enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VS, bits [57:56]

When FEAT_PMUv3_SME is implemented:

SVE mode filtering. Controls counting events in Streaming and Non-streaming SVE modes.

VS	Meaning
0b00	This mechanism has no effect on the filtering of events.
0b01	The PE does not count events in Streaming SVE mode.
0b10	The PE does not count events in Non-streaming SVE mode.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLC, bits [55:54]
When FEAT_PMUv3_TH2 is implemented and n is odd:

Threshold Linking Control. Extends PMEVTYPER<n>_EL0.TC with additional controls for event linking. See PMEVTYPER<n>_EL0.TC.

TLC	Meaning
0b00	Threshold linking disabled.
0b01	Threshold linking enabled. If the threshold condition described by PMEVTYPER<n>_EL0.TC is false, the counter increments by V[n-1]. Otherwise, the counter increments as described by PMEVTYPER<n>_EL0.TC.
0b10	Threshold linking enabled. If the threshold condition described by PMEVTYPER<n>_EL0.TC is true, the counter increments by V[n-1]. Otherwise, the counter does not increment.

All other values are reserved.

See PMEVTYPER<n>_EL0.TC for more information

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [53:44]

Reserved, RES0.

TH, bits [43:32]
When FEAT_PMUv3_TH is implemented:

Threshold value. Provides the unsigned value for the threshold function defined by PMEVTYPER<n>_EL0.TC.

If PMEVTYPER<n>_EL0.{TC, TH} are both zero and either FEAT_PMUv3_TH2 is not implemented or PMEVTYPER<n>_EL0.TLC is also zero, then the threshold function is disabled.

If PMMIR_EL1.THWIDTH is less than 12, then bits PMEVTYPER<n>_EL0.TH[11:UInt(PMMIR_EL1.THWIDTH)] are RES0. This accounts for the behavior when writing a value greater-than-or-equal-to $2^{\text{UInt(PMMIR\_EL1.THWIDTH)}}$.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to '000000000000'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

P, bit [31]

EL1 filtering. Controls counting events in EL1.

P	Meaning
0b0	This mechanism has no effect on filtering of events.
0b1	The PE does not count events in EL1.

If Secure and Non-secure states are implemented, then counting events in Non-secure EL1 is further controlled by PMEVTYPER<n>_EL0.NSK.

If FEAT_RME is implemented, then counting events in Realm EL1 is further controlled by PMEVTYPER<n>_EL0.RLK.

If EL3 is implemented, then counting events in EL3 is further controlled by PMEVTYPER<n>_EL0.M.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

EL0 filtering. Controls counting events in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of events.
0b1	The PE does not count events in EL0.

If Secure and Non-secure states are implemented, then counting events in Non-secure EL0 is further controlled by PMEVTYPER<n>_EL0.NSU.

If FEAT_RME is implemented, then counting events in Realm EL0 is further controlled by PMEVTYPER<n>_EL0.RLU.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting events in Non-secure EL1. If PMEVTYPER<n>_EL0.NSK is not equal to PMEVTYPER<n>_EL0.P, then the PE does not count events in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of events in Non-secure EL1.

NSK	Meaning
0b0	When PMEVTYPER<n>_EL0.P == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.P == 1, the PE does not count events in Non-secure EL1.
0b1	When PMEVTYPER<n>_EL0.P == 0, the PE does not count events in Non-secure EL1. When PMEVTYPER<n>_EL0.P == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 filtering. Controls counting events in Non-secure EL0. If PMEVTYPER<n>_EL0.NSU is not equal to PMEVTYPER<n>_EL0.U, then the PE does not count events in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of events in Non-secure EL0.

NSU	Meaning
0b0	When PMEVTYPER<n>_EL0.U == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.U == 1, the PE does not count events in Non-secure EL0.
0b1	When PMEVTYPER<n>_EL0.U == 0, the PE does not count events in Non-secure EL0. When PMEVTYPER<n>_EL0.U == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 filtering. Controls counting events in EL2.

NSH	Meaning
0b0	The PE does not count events in EL2.
0b1	This mechanism has no effect on filtering of events.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting events in Secure EL2 is further controlled by PMEVTYPER<n>_EL0.SH.

If FEAT_RME is implemented, then counting events in Realm EL2 is further controlled by PMEVTYPER<n>_EL0.RLH.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]

When EL3 is implemented:

EL3 filtering. Controls counting events in EL3. If PMEVTYPER<n>_EL0.M is not equal to PMEVTYPER<n>_EL0.P, then the PE does not count events in EL3. Otherwise, this mechanism has no effect on filtering of events in EL3.

M	Meaning
0b0	When PMEVTYPER<n>_EL0.P == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.P == 1, the PE does not count events in EL3.
0b1	When PMEVTYPER<n>_EL0.P == 0, the PE does not count events in EL3. When PMEVTYPER<n>_EL0.P == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MT, bit [25]
When FEAT_MTPMU is implemented or an IMPLEMENTATION DEFINED multi-threaded PMU extension is implemented:

Multithreading.

MT	Meaning
0b0	Count events only on controlling PE.
0b1	Count events from any PE with the same affinity at level 1 and above as this PE.

Unless otherwise stated:

- If the event counts PE cycles when a stall condition is true, then the counter counts Processor cycles when the stall condition is true for all of these PEs.
- If the event counts PE cycles when any other condition is true, then the counter counts Processor cycles when the condition is true for any of these PEs.
- Otherwise, the event counts by the sum of the count across all of these PEs. See 'Multithreaded implementations' and 'Cycle event counting in multithreaded implementations'.

From Armv8.6, the IMPLEMENTATION DEFINED multi-threaded PMU extension is not permitted, meaning if FEAT_MTPMU is not implemented, this field is RES0. See ID_AA64DFR0_EL1.MTPMU.

This field is ignored by the PE and treated as zero when FEAT_MTPMU is implemented and disabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SH, bit [24]
When EL3 is implemented and FEAT_SEL2 is implemented:

Secure EL2 filtering. Controls counting events in Secure EL2. If PMEVTYPER<n>_EL0.SH is equal to PMEVTYPER<n>_EL0.NSH, then the PE does not count events in Secure EL2. Otherwise, this mechanism has no effect on filtering of events in Secure EL2.

SH	Meaning
0b0	When PMEVTYPER<n>_EL0.NSH == 0, the PE does not count events in Secure EL2. When PMEVTYPER<n>_EL0.NSH == 1, this mechanism has no effect on filtering of events.
0b1	When PMEVTYPER<n>_EL0.NSH == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.NSH == 1, the PE does not count events in Secure EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When Secure EL2 is not implemented, access to this field is **RES0**.

Otherwise:

Reserved, RES0.

T, bit [23]
When FEAT_TME is implemented:

Non-Transactional state filtering field. Controls counting of events in Non-transactional state.

T	Meaning
0b0	This field has no effect on the filtering of events.
0b1	Do not count Attributable events in Non-transactional state.

For each Unattributable event, it is IMPLEMENTATION DEFINED whether the filtering applies.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLK, bit [22]
When FEAT_RME is implemented:

Realm EL1 filtering. Controls counting events in Realm EL1. If PMEVTYPER<n>_EL0.RLK is not equal to PMEVTYPER<n>_EL0.P, then the PE does not count events in Realm EL1. Otherwise, this mechanism has no effect on filtering of events in Realm EL1.

RLK	Meaning
0b0	When PMEVTYPER<n>_EL0.P == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.P == 1, the PE does not count events in Realm EL1.
0b1	When PMEVTYPER<n>_EL0.P == 0, the PE does not count events in Realm EL1. When PMEVTYPER<n>_EL0.P == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]
When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting events in Realm EL0. If PMEVTYPER<n>_EL0.RLU is not equal to PMEVTYPER<n>_EL0.U, then the PE does not count events in Realm EL0. Otherwise, this mechanism has no effect on filtering of events in Realm EL0.

RLU	Meaning
0b0	When PMEVTYPER<n>_EL0.U == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.U == 1, the PE does not count events in Realm EL0.
0b1	When PMEVTYPER<n>_EL0.U == 0, the PE does not count events in Realm EL0. When PMEVTYPER<n>_EL0.U == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]

When FEAT_RME is implemented:

Realm EL2 filtering. Controls counting events in Realm EL2. If PMEVTYPER<n>_EL0.RLH is equal to PMEVTYPER<n>_EL0.NSH, then the PE does not count events in Realm EL2. Otherwise, this mechanism has no effect on filtering of events in Realm EL2.

RLH	Meaning
0b0	When PMEVTYPER<n>_EL0.NSH == 0, the PE does not count events in Realm EL2. When PMEVTYPER<n>_EL0.NSH == 1, this mechanism has no effect on filtering of events.
0b1	When PMEVTYPER<n>_EL0.NSH == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.NSH == 1, the PE does not count events in Realm EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:16]

Reserved, RES0.

evtCount[15:10], bits [15:10]

When FEAT_PMUv3p1 is implemented:

Extension to evtCount[9:0]. For more information, see evtCount[9:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

evtCount[9:0], bits [9:0]

Event to count.

The event number of the event that is counted by event counter PMEVCNTR<n>_EL0.

The ranges of event numbers allocated to each type of event are shown in 'Allocation of the PMU event number space'.

If FEAT_PMUv3p8 is implemented and PMEVTYPER<n>_EL0.evtCount is programmed to an event that is reserved or not supported by the PE, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.

Note

Arm recommends this behavior for all implementations of FEAT_PMUv3.

Otherwise, if PMEVTYPER<n>_EL0.evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range `0x0000` to `0x003F`, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.
- If FEAT_PMUv3p1 is implemented, for the range `0x4000` to `0x403F`, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.
- For other values, it is UNPREDICTABLE what event, if any, is counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is UNKNOWN.

Note

UNPREDICTABLE means the event must not expose privileged information.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMEVTYPER<n>_EL0

PMEVTYPER<n>_EL0 can also be accessed by using PMXEVTYPER_EL0 with PMSELR_EL0.SEL set to n.

If FEAT_FGT is implemented and <n> is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMEVTYPER<n>_EL0 is as follows:

- If <n> is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and <n> is greater than or equal to the number of accessible event counters, then reads and writes of PMEVTYPER<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a `NOP`.
- Accesses to the register behave as if <n> is an UNKNOWN value less-than-or-equal-to the index of the highest accessible event counter.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMEVTYPER<n>_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.

- PMUACR_EL1.P<n> == 0.

Permitted writes of PMEVTYPER<n>_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.{UEN,ER} == {1,1}.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.{UEN,EN}.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible event counters. Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see MDCR_EL2.HPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMEVTYPER<m>_EL0 ; Where m = 0-30
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b11:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1[m] == '0'
then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMEVTYPER_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVTYPER_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVTYPER_EL0[m];
elsif PSTATE.EL == EL3 then
```

```
    X[t, 64] = PMEVTYPER_EL0[m];
```

```
MSR PMEVTYPER<m>_EL0, <Xt> ; Where m = 0-30
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b11:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1[m] == '0'
|| PMUSERENR_EL0.ER == '1') then
        return;
    else
        PMEVTYPER_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVTYPER_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVTYPER_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
```

```
PMEVTYPER_EL0[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMIAR_EL1, Performance Monitors Instruction Address Register

The PMIAR_EL1 characteristics are:

Purpose

Captures the address of the instruction generating a PMU Profiling exception.

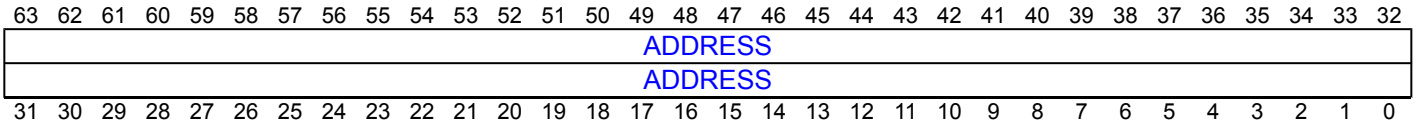
Configuration

This register is present only when FEAT_SEBEP is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMIAR_EL1 are UNDEFINED.

Attributes

PMIAR_EL1 is a 64-bit register.

Field descriptions



ADDRESS, bits [63:0]

Instruction virtual address.

For writes to PMIAR_EL1, PMIAR_EL1.ADDRESS[63:P] is RESS. P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

PMIAR_EL1.ADDRESS[1:0] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMIAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMIAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_SEBEP) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMIAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMIAR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMIAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMIAR_EL1;
```

```
MSR PMIAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_SEBEP) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nPMIAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMIAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMIAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMIAR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMICFILTR_EL0, Performance Monitors Instruction Counter Filter Register

The PMICFILTR_EL0 characteristics are:

Purpose

Configures the Instruction Counter.

Configuration

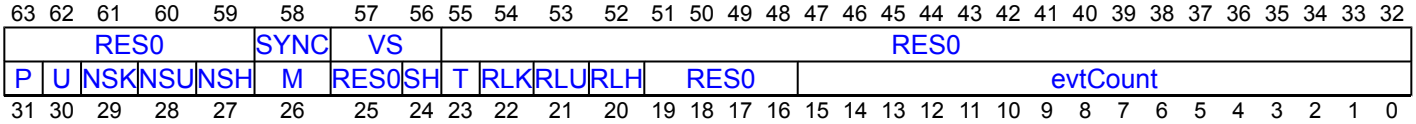
AArch64 System register PMICFILTR_EL0 bits [63:0] are architecturally mapped to External register PMICFILTR_EL0[63:0].

This register is present only when FEAT_PMUv3_ICNTR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMICFILTR_EL0 are UNDEFINED.

Attributes

PMICFILTR_EL0 is a 64-bit register.

Field descriptions



Bits [63:59]

Reserved, RES0.

SYNC, bit [58]

When FEAT_SEBEP is implemented:

Synchronous mode. Controls whether a PMU Profiling exception generated by the counter is synchronous or asynchronous.

SYNC	Meaning
0b0	Asynchronous PMU Profiling exception is enabled.
0b1	Synchronous PMU Profiling exception is enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VS, bits [57:56]

When FEAT_PMUv3_SME is implemented:

SVE mode filtering. Controls counting instructions in Streaming and Non-streaming SVE modes.

VS	Meaning
0b00	This mechanism has no effect on the filtering of instructions.
0b01	The PE does not count instructions in Streaming SVE mode.
0b10	The PE does not count instructions in Non-streaming SVE mode.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [55:32]

Reserved, RES0.

P, bit [31]

EL1 filtering. Controls counting instructions in EL1.

P	Meaning
0b0	This mechanism has no effect on filtering of instructions.
0b1	The PE does not count instructions in EL1.

If Secure and Non-secure states are implemented, then counting instructions in Non-secure EL1 is further controlled by PMICFILTR_EL0.NSK.

If FEAT_RME is implemented, then counting instructions in Realm EL1 is further controlled by PMICFILTR_EL0.RLK.

If EL3 is implemented, then counting instructions in EL3 is further controlled by PMICFILTR_EL0.M.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

EL0 filtering. Controls counting instructions in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of instructions.
0b1	The PE does not count instructions in EL0.

If Secure and Non-secure states are implemented, then counting instructions in Non-secure EL0 is further controlled by PMICFILTR_EL0.NSU.

If FEAT_RME is implemented, then counting instructions in Realm EL0 is further controlled by PMICFILTR_EL0.RLU.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting instructions in Non-secure EL1. If PMICFILTR_EL0.NSK is not equal to PMICFILTR_EL0.P, then the PE does not count instructions in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of instructions in Non-secure EL1.

NSK	Meaning
0b0	When PMICFILTR_EL0.P == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.P == 1, the PE does not count instructions in Non-secure EL1.
0b1	When PMICFILTR_EL0.P == 0, the PE does not count instructions in Non-secure EL1. When PMICFILTR_EL0.P == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 filtering. Controls counting instructions in Non-secure EL0. If PMICFILTR_EL0.NSU is not equal to PMICFILTR_EL0.U, then the PE does not count instructions in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of instructions in Non-secure EL0.

NSU	Meaning
0b0	When PMICFILTR_EL0.U == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.U == 1, the PE does not count instructions in Non-secure EL0.
0b1	When PMICFILTR_EL0.U == 0, the PE does not count instructions in Non-secure EL0. When PMICFILTR_EL0.U == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 filtering. Controls counting instructions in EL2.

NSH	Meaning
0b0	The PE does not count instructions in EL2.
0b1	This mechanism has no effect on filtering of instructions.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting instructions in Secure EL2 is further controlled by PMICFILTR_EL0.SH.

If FEAT_RME is implemented, then counting instructions in Realm EL2 is further controlled by PMICFILTR_EL0.RLH.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]

When EL3 is implemented:

EL3 filtering. Controls counting instructions in EL3. If PMICFILTR_EL0.M is not equal to PMICFILTR_EL0.P, then the PE does not count instructions in EL3. Otherwise, this mechanism has no effect on filtering of instructions in EL3.

M	Meaning
0b0	When PMICFILTR_EL0.P == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.P == 1, the PE does not count instructions in EL3.
0b1	When PMICFILTR_EL0.P == 0, the PE does not count instructions in EL3. When PMICFILTR_EL0.P == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [25]

Reserved, RES0.

SH, bit [24]

When EL3 is implemented and FEAT_SEL2 is implemented:

Secure EL2 filtering. Controls counting instructions in Secure EL2. If PMICFILTR_EL0.SH is equal to PMICFILTR_EL0.NSH, then the PE does not count instructions in Secure EL2. Otherwise, this mechanism has no effect on filtering of instructions in Secure EL2.

SH	Meaning
0b0	When PMICFILTR_EL0.NSH == 0, the PE does not count instructions in Secure EL2. When PMICFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of instructions.
0b1	When PMICFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.NSH == 1, the PE does not count instructions in Secure EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When Secure EL2 is not implemented, access to this field is **RES0**.

Otherwise:

Reserved, RES0.

T, bit [23]

When FEAT_TME is implemented:

Non-Transactional state filtering field. Controls counting of instructions in Non-transactional state.

T	Meaning
0b0	This field has no effect on the filtering of instructions.
0b1	Do not count Attributable instructions in Non-transactional state.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLK, bit [22]

When FEAT_RME is implemented:

Realm EL1 filtering. Controls counting instructions in Realm EL1. If PMICFILTR_EL0.RLK is not equal to PMICFILTR_EL0.P, then the PE does not count instructions in Realm EL1. Otherwise, this mechanism has no effect on filtering of instructions in Realm EL1.

RLK	Meaning
0b0	When PMICFILTR_EL0.P == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.P == 1, the PE does not count instructions in Realm EL1.
0b1	When PMICFILTR_EL0.P == 0, the PE does not count instructions in Realm EL1. When PMICFILTR_EL0.P == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting instructions in Realm EL0. If PMICFILTR_EL0.RLU is not equal to PMICFILTR_EL0.U, then the PE does not count instructions in Realm EL0. Otherwise, this mechanism has no effect on filtering of instructions in Realm EL0.

RLU	Meaning
0b0	When PMICFILTR_EL0.U == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.U == 1, the PE does not count instructions in Realm EL0.
0b1	When PMICFILTR_EL0.U == 0, the PE does not count instructions in Realm EL0. When PMICFILTR_EL0.U == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.

- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]

When FEAT_RME is implemented:

Realm EL2 filtering. Controls counting instructions in Realm EL2. If PMICFILTR_EL0.RLH is equal to PMICFILTR_EL0.NSH, then the PE does not count instructions in Realm EL2. Otherwise, this mechanism has no effect on filtering of instructions in Realm EL2.

RLH	Meaning
0b0	When PMICFILTR_EL0.NSH == 0, the PE does not count instructions in Realm EL2. When PMICFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of instructions.
0b1	When PMICFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.NSH == 1, the PE does not count instructions in Realm EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count.

Reads as 0x0008.

Access to this field is **RO**.

Accessing PMICFILTR_EL0

Permitted reads and writes of PMICFILTR_EL0 are RAZ/WI if all of the following are true:

- PSTATE.EL == EL0.
- PMUACR_EL1.F0 == 0.

Permitted writes of PMICFILTR_EL0 are ignored if all of the following are true:

- PSTATE.EL == EL0.
- PMUSERENR_EL0.IR == 1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMICFILTR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3_ICNTR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.UEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nPMICFILTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.F0 == '0'
then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMICFILTR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMICFILTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMICFILTR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```

```
        X[t, 64] = PMICFILTR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMICFILTR_EL0;
```

```
MSR PMICFILTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3_ICNTR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.UEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nPMICFILTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.F0 == '0'
|| PMUSERENR_EL0.IR == '1') then
        return;
    else
        PMICFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nPMICFILTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMICFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```

```
        PMICFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMICFILTR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMICNTR_EL0, Performance Monitors Instruction Counter Register

The PMICNTR_EL0 characteristics are:

Purpose

If event counting is not prohibited and the instruction counter is enabled, the counter increments for each architecturally-executed instruction, according to the configuration specified by PMICFILTR_EL0.

Configuration

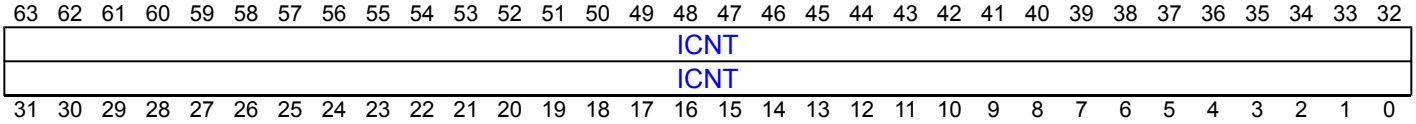
AArch64 System register PMICNTR_EL0 bits [63:0] are architecturally mapped to External register PMICNTR_EL0[63:0].

This register is present only when FEAT_PMUv3_ICNTR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMICNTR_EL0 are UNDEFINED.

Attributes

PMICNTR_EL0 is a 64-bit register.

Field descriptions



ICNT, bits [63:0]

Instruction Counter.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMICNTR_EL0

PMICNTR_EL0 treats permitted reads-as-zero and ignores permitted writes if all of the following are true:

- PSTATE.EL == EL0.
- PMUACR_EL1.F0 == 0.

PMICNTR_EL0 ignores permitted writes if all of the following are true:

- PSTATE.EL == EL0.
- PMUSERENR_EL0.IR == 1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMICNTR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3_ICNTR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.UEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nPMICNTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.F0 == '0'
then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMICNTR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMICNTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMICNTR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```

```
        X[t, 64] = PMICNTR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMICNTR_EL0;
```

```
MSR PMICNTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3_ICNTR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.UEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nPMICNTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.F0 == '0'
|| PMUSERENR_EL0.IR == '1') then
        return;
    else
        PMICNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nPMICNTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMICNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```

```
        PMICNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMICNTR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMICNTSVR_EL1, Performance Monitors Instruction Count Saved Value Register

The PMICNTSVR_EL1 characteristics are:

Purpose

Captures the PMU Instruction counter, PMICNTR_EL0.

Configuration

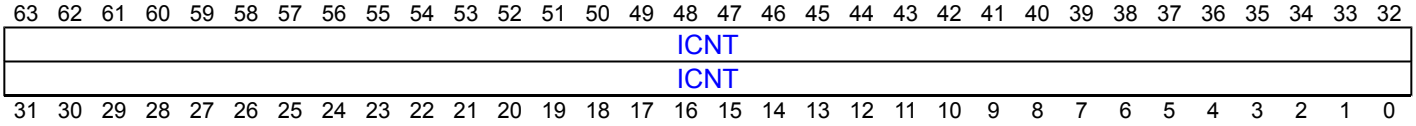
AArch64 System register PMICNTSVR_EL1 bits [63:0] are architecturally mapped to External register PMICNTSVR_EL1[63:0].

This register is present only when FEAT_PMUv3_ICNTR is implemented, FEAT_PMUv3_SS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to PMICNTSVR_EL1 are UNDEFINED.

Attributes

PMICNTSVR_EL1 is a 64-bit register.

Field descriptions



ICNT, bits [63:0]

Sampled Instruction Count. The value of PMICNTR_EL0 at the last successful Capture event.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMICNTSVR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMICNTSVR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1110	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3_ICNTR) && IsFeatureImplemented(FEAT_PMUv3_SS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMSSDATA == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMICNTSVR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMICNTSVR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMICNTSVR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENCLR_EL1, Performance Monitors Interrupt Enable Clear Register

The PMINTENCLR_EL1 characteristics are:

Purpose

Allows software to disable the generation of interrupt requests or, when FEAT_EBEP is implemented, PMU Profiling exceptions on overflows from the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which overflow interrupt requests or PMU Profiling exceptions are enabled.

Configuration

AArch64 System register PMINTENCLR_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[63:0].

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0].

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENSET[31:0].

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to External register PMINTENCLR_EL1[31:0].

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to External register PMINTENSET_EL1[31:0].

AArch64 System register PMINTENCLR_EL1 bits [63:32] are architecturally mapped to External register PMINTENCLR_EL1[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

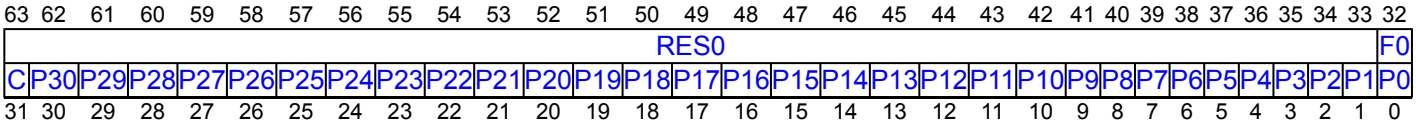
AArch64 System register PMINTENCLR_EL1 bits [63:32] are architecturally mapped to External register PMINTENSET_EL1[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMINTENCLR_EL1 are UNDEFINED.

Attributes

PMINTENCLR_EL1 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enable status.

F0	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - PSTATE.EL != EL3.
  - MDCR_EL3.EnPM2 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL == EL1.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or (HDFGRTR2_EL2.nPMICFILTR_EL0 == 0 and HDFGWTR2_EL2.nPMICFILTR_EL0 == 0).
- Access to this field is **WO/RAZ** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL == EL1.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGRTR2_EL2.nPMICFILTR_EL0 == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL == EL1.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGWTR2_EL2.nPMICFILTR_EL0 == 0.
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

C, bit [31]

Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable status.

C	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing PMINTENCLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMINTENCLR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMINTENCLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMINTENCLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMINTENCLR_EL1;
```

```
MSR PMINTENCLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENCLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENCLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMINTENCLR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENSET_EL1, Performance Monitors Interrupt Enable Set Register

The PMINTENSET_EL1 characteristics are:

Purpose

Allows software to enable the generation of interrupt requests or, when FEAT_EBEP is implemented, PMU Profiling exceptions on overflows from the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which overflow interrupt requests or PMU Profiling exceptions are enabled.

Configuration

AArch64 System register PMINTENSET_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[63:0].

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENSET[31:0].

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0].

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to External register PMINTENSET_EL1[31:0].

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to External register PMINTENCLR_EL1[31:0].

AArch64 System register PMINTENSET_EL1 bits [63:32] are architecturally mapped to External register PMINTENSET_EL1[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

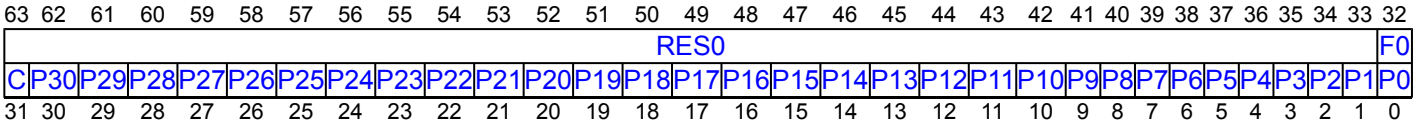
AArch64 System register PMINTENSET_EL1 bits [63:32] are architecturally mapped to External register PMINTENCLR_EL1[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMINTENSET_EL1 are UNDEFINED.

Attributes

PMINTENSET_EL1 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enable status.

F0	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - PSTATE.EL != EL3.
  - MDCR_EL3.EnPM2 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL == EL1.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or (HDFGRTR2_EL2.nPMICFILTR_EL0 == 0 and HDFGWTR2_EL2.nPMICFILTR_EL0 == 0).
- Access to this field is **WO/RAZ** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL == EL1.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGRTR2_EL2.nPMICFILTR_EL0 == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL == EL1.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGWTR2_EL2.nPMICFILTR_EL0 == 0.
- Otherwise, access to this field is **W1S**.

Otherwise:

Reserved, RES0.

C, bit [31]

Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable status.

C	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing PMINTENSET_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMINTENSET_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMINTENSET_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMINTENSET_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMINTENSET_EL1;
```

```
MSR PMINTENSET_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENSET_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENSET_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMINTENSET_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMMIR_EL1, Performance Monitors Machine Identification Register

The PMMIR_EL1 characteristics are:

Purpose

Describes Performance Monitors parameters specific to the implementation to software.

Configuration

AArch64 System register PMMIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMMIR[31:0].

AArch64 System register PMMIR_EL1 bits [31:0] are architecturally mapped to External register PMMIR[31:0] when FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p4 is implemented.

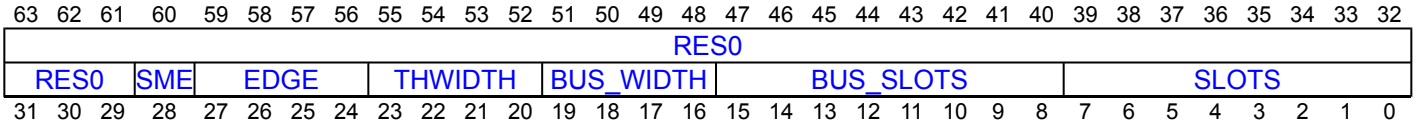
AArch64 System register PMMIR_EL1 bits [63:32] are architecturally mapped to External register PMMIR[63:32] when FEAT_PMUv3_EXT is implemented, FEAT_PMUv3p4 is implemented, and (FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented).

This register is present only when FEAT_PMUv3p4 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMMIR_EL1 are UNDEFINED.

Attributes

PMMIR_EL1 is a 64-bit register.

Field descriptions



Bits [63:29]

Reserved, RES0.

SME, bit [28]

PMUv3 for SME. Adds support for the Streaming SVE mode filter.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SME	Meaning
0b0	Streaming SVE mode filter not implemented.
0b1	Adds support for the Streaming SVE mode filter.

All other values are reserved.

FEAT_PMUv3_SME implements the functionality identified by the value 1.

Access to this field is **RO**.

EDGE, bits [27:24]

PMU event edge detection. With PMMIR_EL1.THWIDTH, indicates implementation of event counter thresholding features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EDGE	Meaning
0b0000	FEAT_PMUv3_EDGE is not implemented.
0b0001	FEAT_PMUv3_EDGE is implemented.
0b0010	As 0b0001, and adds support for threshold value linking between a pair of counters.

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, the only permitted value is 0b0000.

FEAT_PMUv3_EDGE implements the functionality identified by the value 0b0001.

FEAT_PMUv3_TH2 implements the functionality identified by the value 0b0010.

Access to this field is **RO**.

THWIDTH, bits [23:20]

PMEVTYPER<n>_EL0.TH width. Indicates implementation of the FEAT_PMUv3_TH feature, and, if implemented, the size of the PMEVTYPER<n>_EL0.TH field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

THWIDTH	Meaning
0b0000	FEAT_PMUv3_TH is not implemented.
0b0001	1 bit. PMEVTYPER<n>_EL0.TH[11:1] are RES0.
0b0010	2 bits. PMEVTYPER<n>_EL0.TH[11:2] are RES0.
0b0011	3 bits. PMEVTYPER<n>_EL0.TH[11:3] are RES0.
0b0100	4 bits. PMEVTYPER<n>_EL0.TH[11:4] are RES0.
0b0101	5 bits. PMEVTYPER<n>_EL0.TH[11:5] are RES0.
0b0110	6 bits. PMEVTYPER<n>_EL0.TH[11:6] are RES0.
0b0111	7 bits. PMEVTYPER<n>_EL0.TH[11:7] are RES0.
0b1000	8 bits. PMEVTYPER<n>_EL0.TH[11:8] are RES0.
0b1001	9 bits. PMEVTYPER<n>_EL0.TH[11:9] are RES0.
0b1010	10 bits. PMEVTYPER<n>_EL0.TH[11:10] are RES0.
0b1011	11 bits. PMEVTYPER<n>_EL0.TH[11] is RES0.
0b1100	12 bits.

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, this field is zero.

Otherwise, the largest value that can be written to PMEVTYPER<n>_EL0.TH is $2^{(\text{PMMIR\_EL1.THWIDTH})}$ minus one.

Access to this field is **RO**.

BUS_WIDTH, bits [19:16]

Bus width. Indicates the number of bytes each BUS_ACCESS event relates to. Encoded as Log_2(number of bytes), plus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BUS_WIDTH	Meaning
0b0000	The information is not available.
0b0011	Four bytes.
0b0100	8 bytes.
0b0101	16 bytes.
0b0110	32 bytes.
0b0111	64 bytes.
0b1000	128 bytes.
0b1001	256 bytes.
0b1010	512 bytes.
0b1011	1024 bytes.
0b1100	2048 bytes.

All other values are reserved.

Each transfer is up to this number of bytes. An access might be smaller than the bus width.

When this field is nonzero, each access counted by BUS_ACCESS is at most BUS_WIDTH bytes. An implementation might treat a wide bus as multiple narrower buses, such that a wide access on the bus increments the BUS_ACCESS counter by more than one.

Access to this field is **RO**.

BUS_SLOTS, bits [15:8]

Bus count. The largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle.

When this field is nonzero, the largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle is BUS_SLOTS.

If the bus count information is not available, this field will read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SLOTS, bits [7:0]

Operation width. The largest value by which the STALL_SLOT event might increment in a single cycle. If the STALL_SLOT event is not implemented, this field might read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMMIR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMMIR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b110

```
if !(IsFeatureImplemented(FEAT_PMUv3p4) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMMIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMMIR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMMIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMMIR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVSCLR_EL0, Performance Monitors Overflow Flag Status Clear Register

The PMOVSCLR_EL0 characteristics are:

Purpose

Allows software to clear the unsigned overflow flags for the following counters to 0:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows the current unsigned overflow flag values.

Configuration

AArch64 System register PMOVSCLR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMOVSSET_EL0[63:0].

AArch64 System register PMOVSCLR_EL0 bits [63:0] are architecturally mapped to External register PMOVS[63:0].

AArch64 System register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSR[31:0].

AArch64 System register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSSET[31:0].

AArch64 System register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to External register PMOVSCLR_EL0[31:0].

AArch64 System register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to External register PMOVSSET_EL0[31:0].

AArch64 System register PMOVSCLR_EL0 bits [63:32] are architecturally mapped to External register PMOVSCLR_EL0[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

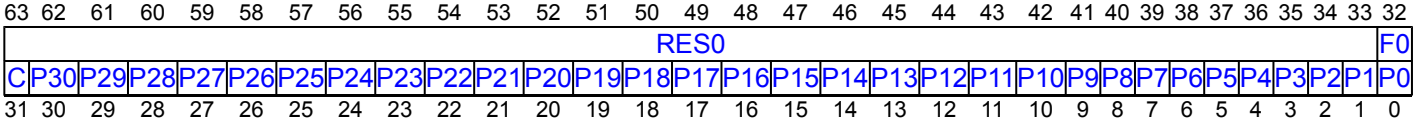
AArch64 System register PMOVSCLR_EL0 bits [63:32] are architecturally mapped to External register PMOVSSET_EL0[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMOVSCLR_EL0 are UNDEFINED.

Attributes

PMOVSCLR_EL0 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Unsigned overflow flag for PMICNTR_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMICNTR_EL0 to 0. On reads, returns the unsigned overflow flag for PMICNTR_EL0 overflow status.

F0	Meaning
0b0	PMICNTR_EL0 has not overflowed.
0b1	PMICNTR_EL0 has overflowed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - PSTATE.EL != EL3.
  - MDCR_EL3.EnPM2 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 0 or PMUACR_EL1.F0 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or (HDFGRTR2_EL2.nPMICFILTR_EL0 == 0 and HDFGWTR2_EL2.nPMICFILTR_EL0 == 0).
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL == EL0.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGRTR2_EL2.nPMICFILTR_EL0 == 0.
  - PMUSERENR_EL0.IR == 1.
- Access to this field is **WO/RAZ** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGRTR2_EL2.nPMICFILTR_EL0 == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGWTR2_EL2.nPMICFILTR_EL0 == 0.
- Access to this field is **RO** if all the following are true:
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.IR == 1.
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

C, bit [31]

Unsigned overflow flag for PMCCNTR_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMCCNTR_EL0 to 0. On reads, returns the unsigned overflow flag for PMCCNTR_EL0 overflow status.

C	Meaning
0b0	PMCCNTR_EL0 has not overflowed.
0b1	PMCCNTR_EL0 has overflowed.

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.

- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.C == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,CR] == 0b11.
- Otherwise, access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

Unsigned overflow flag for PMEVCNTR<m>_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMEVCNTR<m>_EL0 to 0. On reads, returns the unsigned overflow flag for PMEVCNTR<m>_EL0 overflow status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 has not overflowed.
0b1	PMEVCNTR<m>_EL0 has overflowed.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>_EL0[31:0] or unsigned overflow of PMEVCNTR<m>_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.P<m> == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,ER] == 0b11.
- Otherwise, access to this field is **W1C**.

Accessing PMOVSCLR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMOVSCLR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSCLR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSCLR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSCLR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMOVSCLR_EL0;
```

```
MSR PMOVSCLR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMOVSCLR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVSSET_EL0, Performance Monitors Overflow Flag Status Set Register

The PMOVSSET_EL0 characteristics are:

Purpose

Allows software to set the unsigned overflow flags for the following counters to 1:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows the current unsigned overflow flag values.

Configuration

AArch64 System register PMOVSSET_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMOVSCLR_EL0[63:0].

AArch64 System register PMOVSSET_EL0 bits [63:0] are architecturally mapped to External register PMOVS[63:0].

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSSET[31:0].

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSR[31:0].

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to External register PMOVSCLR_EL0[31:0].

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to External register PMOVSSET_EL0[31:0].

AArch64 System register PMOVSSET_EL0 bits [63:32] are architecturally mapped to External register PMOVSCLR_EL0[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

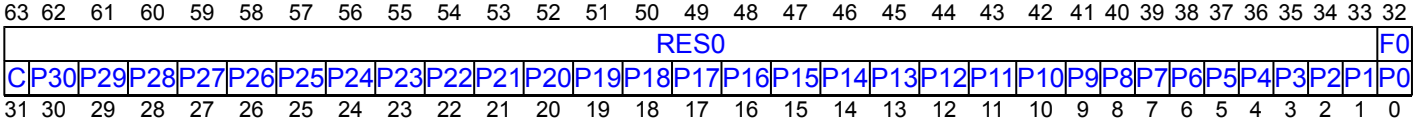
AArch64 System register PMOVSSET_EL0 bits [63:32] are architecturally mapped to External register PMOVSSET_EL0[63:32] when FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented.

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMOVSSET_EL0 are UNDEFINED.

Attributes

PMOVSSET_EL0 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Unsigned overflow flag for PMICNTR_EL0 set. On writes, allows software to set the unsigned overflow flag for PMICNTR_EL0 to 1. On reads, returns the unsigned overflow flag for PMICNTR_EL0 overflow status.

F0	Meaning
0b0	PMICNTR_EL0 has not overflowed.
0b1	PMICNTR_EL0 has overflowed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - PSTATE.EL != EL3.
  - MDCR_EL3.EnPM2 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 0 or PMUACR_EL1.F0 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or (HDFGRTR2_EL2.nPMICFILTR_EL0 == 0 and HDFGWTR2_EL2.nPMICFILTR_EL0 == 0).
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL == EL0.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGRTR2_EL2.nPMICFILTR_EL0 == 0.
  - PMUSERENR_EL0.IR == 1.
- Access to this field is **WO/RAZ** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGRTR2_EL2.nPMICFILTR_EL0 == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGWTR2_EL2.nPMICFILTR_EL0 == 0.
- Access to this field is **RO** if all the following are true:
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.IR == 1.
- Otherwise, access to this field is **W1S**.

Otherwise:

Reserved, RES0.

C, bit [31]

Unsigned overflow flag for PMCCNTR_EL0 set. On writes, allows software to set the unsigned overflow flag for PMCCNTR_EL0 to 1. On reads, returns the unsigned overflow flag for PMCCNTR_EL0 overflow status.

C	Meaning
0b0	PMCCNTR_EL0 has not overflowed.
0b1	PMCCNTR_EL0 has overflowed.

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.

- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.C == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,CR] == 0b11.
- Otherwise, access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

Unsigned overflow flag for PMEVCNTR<m>_EL0 set. On writes, allows software to set the unsigned overflow flag for PMEVCNTR<m>_EL0 to 1. On reads, returns the unsigned overflow flag for PMEVCNTR<m>_EL0 overflow status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 has not overflowed.
0b1	PMEVCNTR<m>_EL0 has overflowed.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>_EL0[31:0] or unsigned overflow of PMEVCNTR<m>_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.P<m> == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,ER] == 0b11.
- Otherwise, access to this field is **W1S**.

Accessing PMOVSSET_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMOVSSET_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1110	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSSET_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSSET_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSSET_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMOVSSET_EL0;
```

```
MSR PMOVSSET_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1110	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMOVSSET_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSCR_EL1, Statistical Profiling Control Register (EL1)

The PMSCR_EL1 characteristics are:

Purpose

Provides EL1 controls for Statistical Profiling.

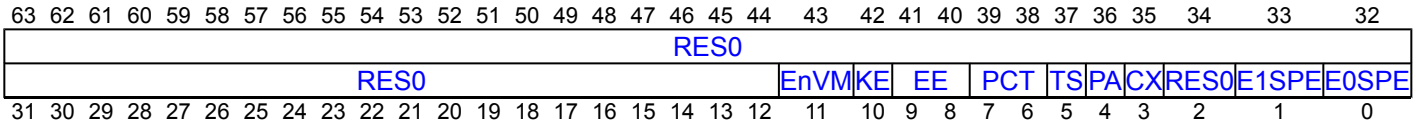
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSCR_EL1 are UNDEFINED.

Attributes

PMSCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:12]

Reserved, RES0.

EnVM, bit [11]

When FEAT_SPE_nVM is implemented and FEAT_NV is implemented:

Reserved for software use in nested virtualization. See also PMSCR_EL2.EnVM.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

KE, bit [10]

When FEAT_SPE_EXC is implemented:

Kernel exception enable for SPE Profiling exceptions taken to EL1.

KE	Meaning
0b0	SPE Profiling exceptions taken to EL1 are always masked at EL1.
0b1	Enabled SPE Profiling exceptions taken to EL1 are masked at EL1 when PSTATE.PM is 1 and unmasked when PSTATE.PM is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bits [9:8]
When FEAT_SPE_EXC is implemented:

Exception Enable.

EE	Meaning	Applies when
0b00	Disabled. SPE Profiling exceptions for EL1 are disabled. All of the following apply: • Unless enabled by a higher Exception level, SPE Profiling exceptions are not generated. • PMBSR_EL1.S drives the interrupt request signal **PMBIRQ**. • Accesses to PMBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1.	
0b01	Reserved for software use in nested virtualization. Behaves as 0b00 for the purpose of controlling the SPE Profiling exception and interrupt request signal **PMBIRQ**, and as 0b11 for the purpose of accesses to PMBSR_EL1.	When FEAT_NV is implemented
0b10	Reserved for software use in nested virtualization. Behaves as 0b11 for the purposes of controlling the SPE Profiling exception and interrupt request signal **PMBIRQ**, and accesses to PMBSR_EL1.	When FEAT_NV is implemented
0b11	Enabled. SPE Profiling exceptions for EL1 are enabled, as follows: • All Profiling Buffer management events are recorded in PMBSR_EL1, unless they are configured to be recorded in PMBSR_EL3 by MDCR_EL3.PMSEE or PMBSR_EL2 by PMSCR_EL2.EE. • SPE Profiling exceptions are generated and taken to EL1 when unmasked and PMBSR_EL1.S is 1, unless the Effective value of HCR_EL2.TGE is 1, in which case the exception is taken to EL2. • The interrupt request signal **PMBIRQ** is not asserted.	

For more information on the values reserved for software use in nested virtualization, see PMSCR_EL2.EE.

If the Effective value of PMSCR_EL2.EE is 0b00, then the Effective value of PMSCR_EL1.EE is 0b00.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PCT, bits [7:6]
When EL2 is implemented:

Physical Timestamp. If timestamp sampling is enabled and the Profiling Buffer owning Exception level is EL1, requests which timestamp counter value is collected.

If FEAT_ECV is implemented, this is a two-bit field as shown. Otherwise, bit[7] is RES0.

PCT	Meaning	Applies when
0b00	Virtual timestamp. The collected timestamp is the physical counter minus the value of CNTVOFF_EL2.	
0b01	Physical timestamp. The collected timestamp is the physical counter.	
0b11	Guest physical timestamp. The collected timestamp is the physical counter minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • SCR_EL3.ECVEn == 0. • CNTHCTL_EL2.ECV == 0. • FEAT_ECV_POFF is not implemented.	When FEAT_ECV is implemented

When EL2 is implemented, all of the following apply:

- If the Profiling Buffer owning Exception level is EL1 and EL2 is enabled in the owning Security state, then the value of PMSCR_EL2.PCT might override or modify the meaning of this field.
- If the Profiling Buffer owning Exception level is EL2, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Physical Timestamp. Reserved. This field reads as 0b01 and ignores writes. Software should treat this field as UNK/SBZP.

When EL2 is not implemented, the Effective values of CNTVOFF_EL2 and CNTPOFF_EL2 are zero, meaning the virtual counter and physical counter have the same value.

TS, bit [5]

Timestamp sample enable. Enables recording of a Timestamp packet when the owning Exception level is EL1.

TS	Meaning
0b0	Timestamp packet recording disabled.
0b1	Timestamp packet recording enabled.

If the Profiling Buffer owning Exception level is EL2, then this field is ignored by the PE. For more information, see 'Controlling the data that is collected'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PA, bit [4]

Physical Address sample enable.

PA	Meaning
0b0	Physical addresses are not collected.
0b1	Physical addresses are collected.

When EL2 is implemented, all of the following apply:

- If the Profiling Buffer owning Exception level is EL1, then this field is combined with the Effective value PMSCR_EL2.PA to determine whether Physical addresses are collected.
- If the Profiling Buffer owning Exception level is EL2, then this field is ignored by the PE.

For more information, see 'Controlling the data that is collected'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CX, bit [3]

CONTEXTIDR_EL1 sample enable. Enables recording of a Context packet containing the value of CONTEXTIDR_EL1.

CX	Meaning
0b0	CONTEXTIDR_EL1 recording disabled.
0b1	CONTEXTIDR_EL1 recording enabled.

The PE ignores the value of this field and CONTEXTIDR_EL1 is not recorded when any of the following apply:

- The sampled operation executes at EL2.
- The sampled operation executes at EL0 and the Effective value of HCR_EL2.TGE is 1.

For more information, see 'Controlling the data that is collected'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E1SPE, bit [1]

EL1 Statistical Profiling Enable.

E1SPE	Meaning
0b0	Sampling disabled at EL1.
0b1	Sampling enabled at EL1.

If the Effective value of HCR_EL2.TGE is 1, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E0SPE, bit [0]

EL0 Statistical Profiling Enable. Controls sampling at EL0 when HCR_EL2.TGE == 0 or if EL2 is disabled or not implemented.

E0SPE	Meaning
0b0	Sampling disabled at EL0.
0b1	Sampling enabled at EL0.

If the Effective value of HCR_EL2.TGE is 1, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x828];
    else
        X[t, 64] = PMSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PMSCR_EL2;
    else
        X[t, 64] = PMSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSCR_EL1;
```

```
MSR PMSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x828] = X[t, 64];
    else
        PMSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PMSCR_EL2 = X[t, 64];
    else
        PMSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, PMSCR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x828];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMSCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = PMSCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR PMSCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x828] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        PMSCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSCR_EL2, Statistical Profiling Control Register (EL2)

The PMSCR_EL2 characteristics are:

Purpose

Provides EL2 controls for Statistical Profiling.

Configuration

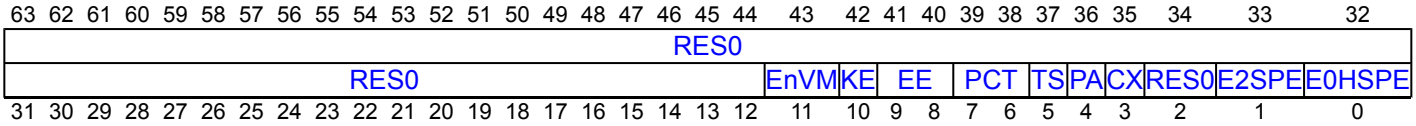
This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSCR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

PMSCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:12]

Reserved, RES0.

EnVM, bit [11]

When FEAT_SPE_nVM is implemented:

Enable use of physical address Profiling Buffer pointers.

EnVM	Meaning
0b0	Use of physical address Profiling Buffer pointers is disabled. The PE behaves as if PMBLIMITR_EL1.nVM is 0.
0b1	Use of physical address Profiling Buffer pointers is permitted.

If EL2 is disabled in the owning Security state, or the Profiling Buffer owning Exception level is EL2, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

KE, bit [10]

When FEAT_SPE_EXC is implemented:

Kernel exception enable for SPE Profiling exceptions taken to EL2.

KE	Meaning
0b0	SPE Profiling exceptions taken to EL2 are always masked at EL2.
0b1	Enabled SPE Profiling exceptions taken to EL2 are masked at EL2 when PSTATE.PM is 1 and unmasked when PSTATE.PM is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bits [9:8]
When FEAT_SPE_EXC is implemented:

Exception Enable.

EE	Meaning
0b00	Disabled. SPE Profiling exceptions for EL2 and EL1 are disabled. All of the following apply: • No Profiling Buffer management events are recorded in PMBSR_EL2. • Unless enabled by a higher Exception level, SPE Profiling exceptions are not generated. • PMBSR_EL1.S drives the interrupt request signal **PMBIRQ**. • Accesses to PMBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1 and accesses to PMBSR_EL1 at EL2 ignore the value of HCR_EL2.E2H.
0b01	Delegated. SPE Profiling exceptions for EL2 are disabled, but might be enabled for EL1 by PMSCR_EL1.EE. All of the following apply: • No Profiling Buffer management events are recorded in PMBSR_EL2. • PMBSR_EL2.S is ignored and SPE Profiling exceptions are not taken to EL2, other than for the case when the Effective value of HCR_EL2.TGE is 1.
0b10	Enabled. SPE Profiling exceptions for EL2 are enabled for Profiling Buffer management events targeting EL2, as follows: • Profiling Buffer management events due to a fault on a write to the Profiling Buffer that would generate a Data Abort exception taken to EL2 if generated by a store instruction executed at the owning Exception level are recorded in PMBSR_EL2, unless they are configured to be recorded in PMBSR_EL3 by MDCR_EL3.PMSEE. If the Profiling Buffer owning Exception level is EL2, then this means any fault on a write to the Profiling Buffer. If the Profiling Buffer owning Exception level is EL1, then this means any of the following faults on a write to the Profiling Buffer: ◦ Stage 2 faults. ◦ If HCR_EL2.TEA is 1, synchronous External aborts. ◦ If HCR_EL2.GPF is 1, Granule Protection Faults (GPFs). • Profiling Buffer management events due to Granule Protection Check faults other than GPFs on a write to the Profiling Buffer are recorded in PMBSR_EL2, unless they are configured to be recorded in PMBSR_EL3 by MDCR_EL3.PMSEE. • SPE Profiling exceptions are generated and taken to EL2 when unmasked and PMBSR_EL2.S is 1.
0b11	Trap all. SPE Profiling exceptions for EL2 are enabled for all Profiling Buffer management events, as follows: • All Profiling Buffer management events are recorded in PMBSR_EL2, unless they are configured to be recorded in PMBSR_EL3 by MDCR_EL3.PMSEE. • SPE Profiling exceptions are generated and taken to EL2 when unmasked and PMBSR_EL2.S is 1.

If the Effective value of MDCR_EL3.PMSEE is 0b00, then the Effective value of PMSCR_EL2.EE is 0b00. Otherwise, if EL2 is not implemented or the Effective value of SCR_EL3.{NS, EEL2} is {0, 0}, then the Effective value of PMSCR_EL2.EE is 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PCT, bits [7:6]

Physical Timestamp. If timestamp sampling is enabled, determines which counter is collected. The behavior depends on the Profiling Buffer owning Exception level.

If FEAT_ECV is implemented, this is a two-bit field as shown. Otherwise, bit[7] is RES0.

PCT	Meaning	Applies when
0b00	Virtual timestamp. The collected timestamp is the physical counter minus a virtual offset. If the Profiling Buffer owning Exception level is EL2 and any of the following are true, then the virtual offset is zero: • The sampled operation executed at EL2 and the Effective value of HCR_EL2.E2H is 1. • The sampled operation executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}. Otherwise, the virtual offset is the value of CNTVOFF_EL2.	
0b01	If the Profiling Buffer owning Exception level is EL1, then the timestamp value is selected by PMSCR_EL1.PCT. Otherwise, physical timestamp. The collected timestamp is the physical counter.	
0b11	If the Profiling Buffer owning Exception level is EL1 and PMSCR_EL1.PCT is 0b00, then guest virtual timestamp. The collected timestamp is the physical counter minus the value of CNTVOFF_EL2. Otherwise, guest physical timestamp. The collected timestamp is the physical counter minus a physical offset. If any of the following are true, then the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • SCR_EL3.ECVEn is 0. • CNTHCTL_EL2.ECV is 0. • FEAT_ECV_POFF is not implemented.	When FEAT_ECV is implemented

All other values are reserved.

If EL2 is not implemented or EL2 is disabled in the current Security state, then the Effective value of this field is 0b01, other than for a direct read of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TS, bit [5]

Timestamp sample enable. Enables recording of a Timestamp packet when the owning Exception level is EL2.

TS	Meaning
0b0	Timestamp packet recording disabled.
0b1	Timestamp packet recording enabled.

If the Profiling Buffer owning Exception level is EL1, then this field is ignored by the PE. For more information, see 'Controlling the data that is collected'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PA, bit [4]

Physical Address Sample Enable.

PA	Meaning
0b0	Physical addresses are not collected.
0b1	Physical addresses are collected.

If the Profiling Buffer owning Exception level is EL1, and EL2 is enabled in the owning Security state, then this field is combined with PMSCR_EL1.PA to determine whether physical addresses are collected.

If EL2 is not implemented or EL2 is disabled in the owning Security state, then the Effective value of PMSCR_EL1.PA is 1.

For more information, see 'Controlling the data that is collected'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CX, bit [3]

CONTEXTIDR_EL2 sample enable. Enables recording of a Context packet containing the value of CONTEXTIDR_EL2.

CX	Meaning
0b0	CONTEXTIDR_EL2 recording disabled.
0b1	CONTEXTIDR_EL2 recording enabled.

The PE ignores the value of this field and CONTEXTIDR_EL2 is not recorded when EL2 is not implemented or EL2 is disabled in the current Security state.

For more information, see 'Controlling the data that is collected'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E2SPE, bit [1]

EL2 Statistical Profiling Enable.

E2SPE	Meaning
0b0	Sampling disabled at EL2.
0b1	Sampling enabled at EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MDCR_EL2.E2PB != 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

E0HSPE, bit [0]

EL0 Statistical Profiling Enable.

E0HSPE	Meaning
0b0	Sampling disabled at EL0.
0b1	Sampling enabled at EL0.

If the Effective value of HCR_EL2.TGE is 0, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MDCR_EL2.E2PB != 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing PMSCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSCR_EL2;
```

```
MSR PMSCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSCR_EL2 = X[t, 64];
```

MRS <Xt>, PMSCR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x828];
    else
        X[t, 64] = PMSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PMSCR_EL2;
    else
        X[t, 64] = PMSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSCR_EL1;
```

```
MSR PMSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x828] = X[t, 64];
    else
        PMSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PMSCR_EL2 = X[t, 64];
    else
        PMSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSDSFR_EL1, Sampling Data Source Filter Register

The PMSDSFR_EL1 characteristics are:

Purpose

Controls sample filtering by Data Source.

Configuration

This register is present only when FEAT_SPE_FDS is implemented. Otherwise, direct accesses to PMSDSFR_EL1 are UNDEFINED.

Attributes

PMSDSFR_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
S63	S62	S61	S60	S59	S58	S57	S56	S55	S54	S53	S52	S51	S50	S49	S48	S47	S46	S45	S44	S43	S42	S41	S40	S39	S38	S37	S36	S35	S34	S33
S31	S30	S29	S28	S27	S26	S25	S24	S23	S22	S21	S20	S19	S18	S17	S16	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

S<m>, bit [m], for m = 63 to 0

When filtering on Data Source <m> is supported:

S[<m>] is the Data Source filter for IMPLEMENTATION DEFINED Data Source <m>.

S<m>	Meaning
0b0	If PMSFCR_EL1.FDS is 1, do not record load operations that have bits [5:0] of the Data Source packet set to <m>.
0b1	Load operations with Data Source <m> are unaffected by PMSFCR_EL1.FDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Accessing PMSDSFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSDSFR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b100

```
if !IsFeatureImplemented(FEAT_SPE_FDS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS3 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMSDSFR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS3 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x858];
    else
        X[t, 64] = PMSDSFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS3 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS3 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSDSFR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSDSFR_EL1;
```

```
MSR PMSDSFR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b100

```
if !IsFeatureImplemented(FEAT_SPE_FDS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS3 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nPMSDSFR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS3 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x858] = X[t, 64];
    else
        PMSDSFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS3 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS3 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSDSFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSDSFR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSELR_EL0, Performance Monitors Event Counter Selection Register

The PMSELR_EL0 characteristics are:

Purpose

Selects the current event counter PMEVCNTR<n>_EL0 or the cycle counter PMCCNTR_EL0.

Used in conjunction with PMXEVTYPER_EL0 to determine the event that increments a selected counter, and the modes and states in which the selected counter increments.

Used in conjunction with PMXEVCNTR_EL0 to determine the value of a selected counter.

Configuration

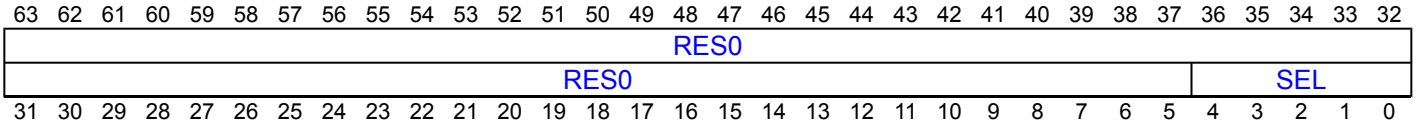
AArch64 System register PMSELR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMSELR[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMSELR_EL0 are UNDEFINED.

Attributes

PMSELR_EL0 is a 64-bit register.

Field descriptions



Bits [63:5]

Reserved, RES0.

SEL, bits [4:0]

Event counter select. Selects the counter accessed by subsequent accesses to PMXEVTYPER_EL0 and PMXEVCNTR_EL0.

SEL	Meaning
0b00000..0b11110	Select event counter PMEVCNTR<n>_EL0, where n is the value of this field: • MRS and MSR of PMXEVTYPER_EL0 access PMEVTYPER<n>_EL0. • MRS and MSR of PMXEVCNTR_EL0 access PMEVCNTR<n>_EL0.
0b11111	Select the cycle counter, PMCCNTR_EL0: • MRS and MSR of PMXEVTYPER_EL0 access PMCCFILTR_EL0. • MRS and MSR of PMXEVCNTR_EL0 are CONSTRAINED UNPREDICTABLE. For more information, see PMXEVCNTR_EL0.

If FEAT_FGT is not implemented and this field is set to a value greater than or equal to the number of implemented counters, but not equal to 31, then direct reads of this field return an UNKNOWN value.

For more information about the results of accesses to the event counters, including when PMSELR_EL0.SEL is set to the index of an unimplemented or inaccessible event counter, see PMXEVTYPER_EL0 and PMXEVCNTR_EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSELR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSELR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSELR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSELR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSELR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSELR_EL0;
```

```
MSR PMSELR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSELR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSEVFR_EL1, Sampling Event Filter Register

The PMSEVFR_EL1 characteristics are:

Purpose

Controls sample filtering by events. The overall filter is the logical AND of these filters. For example, if PMSEVFR_EL1.E[3] and PMSEVFR_EL1.E[5] are both set to 1, only samples that have both event 3 (Level 1 unified or data cache refill) and event 5 (TLB walk) set to 1 are recorded.

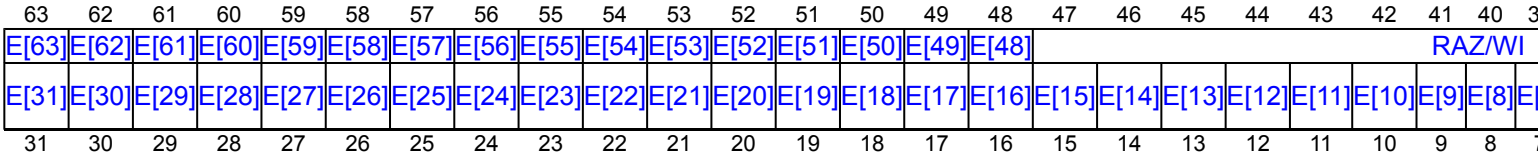
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSEVFR_EL1 are UNDEFINED.

Attributes

PMSEVFR_EL1 is a 64-bit register.

Field descriptions



E[63], bit [63]

When event 63 is implemented and filtering on event 63 is supported:

Filter on IMPLEMENTATION DEFINED event 63.

E[63]	Meaning
0b0	Event 63 is ignored.
0b1	Do not record samples that have event 63 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[62], bit [62]

When event 62 is implemented and filtering on event 62 is supported:

Filter on IMPLEMENTATION DEFINED event 62.

E[62]	Meaning
0b0	Event 62 is ignored.
0b1	Do not record samples that have event 62 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[61], bit [61]

When event 61 is implemented and filtering on event 61 is supported:

Filter on IMPLEMENTATION DEFINED event 61.

E[61]	Meaning
0b0	Event 61 is ignored.
0b1	Do not record samples that have event 61 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[60], bit [60]

When event 60 is implemented and filtering on event 60 is supported:

Filter on IMPLEMENTATION DEFINED event 60.

E[60]	Meaning
0b0	Event 60 is ignored.
0b1	Do not record samples that have event 60 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[59], bit [59]

When event 59 is implemented and filtering on event 59 is supported:

Filter on IMPLEMENTATION DEFINED event 59.

E[59]	Meaning
0b0	Event 59 is ignored.
0b1	Do not record samples that have event 59 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[58], bit [58]

When event 58 is implemented and filtering on event 58 is supported:

Filter on IMPLEMENTATION DEFINED event 58.

E[58]	Meaning
0b0	Event 58 is ignored.
0b1	Do not record samples that have event 58 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[57], bit [57]

When event 57 is implemented and filtering on event 57 is supported:

Filter on IMPLEMENTATION DEFINED event 57.

E[57]	Meaning
0b0	Event 57 is ignored.
0b1	Do not record samples that have event 57 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[56], bit [56]

When event 56 is implemented and filtering on event 56 is supported:

Filter on IMPLEMENTATION DEFINED event 56.

E[56]	Meaning
0b0	Event 56 is ignored.
0b1	Do not record samples that have event 56 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[55], bit [55]

When event 55 is implemented and filtering on event 55 is supported:

Filter on IMPLEMENTATION DEFINED event 55.

E[55]	Meaning
0b0	Event 55 is ignored.
0b1	Do not record samples that have event 55 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[54], bit [54]

When event 54 is implemented and filtering on event 54 is supported:

Filter on IMPLEMENTATION DEFINED event 54.

E[54]	Meaning
0b0	Event 54 is ignored.
0b1	Do not record samples that have event 54 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[53], bit [53]
When event 53 is implemented and filtering on event 53 is supported:

Filter on IMPLEMENTATION DEFINED event 53.

E[53]	Meaning
0b0	Event 53 is ignored.
0b1	Do not record samples that have event 53 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[52], bit [52]
When event 52 is implemented and filtering on event 52 is supported:

Filter on IMPLEMENTATION DEFINED event 52.

E[52]	Meaning
0b0	Event 52 is ignored.
0b1	Do not record samples that have event 52 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[51], bit [51]
When event 51 is implemented and filtering on event 51 is supported:

Filter on IMPLEMENTATION DEFINED event 51.

E[51]	Meaning
0b0	Event 51 is ignored.
0b1	Do not record samples that have event 51 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[50], bit [50]

When event 50 is implemented and filtering on event 50 is supported:

Filter on IMPLEMENTATION DEFINED event 50.

E[50]	Meaning
0b0	Event 50 is ignored.
0b1	Do not record samples that have event 50 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[49], bit [49]

When event 49 is implemented and filtering on event 49 is supported:

Filter on IMPLEMENTATION DEFINED event 49.

E[49]	Meaning
0b0	Event 49 is ignored.
0b1	Do not record samples that have event 49 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[48], bit [48]

When event 48 is implemented and filtering on event 48 is supported:

Filter on IMPLEMENTATION DEFINED event 48.

E[48]	Meaning
0b0	Event 48 is ignored.
0b1	Do not record samples that have event 48 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bits [47:32]

Reserved, RAZ/WI.

E[31], bit [31]

When FEAT_SPEv1p4 is not implemented, event 31 is implemented, and filtering on event 31 is supported:

Filter on IMPLEMENTATION DEFINED event 31.

E[31]	Meaning
0b0	Event 31 is ignored.
0b1	Do not record samples that have event 31 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[30], bit [30]

When FEAT_SPEv1p4 is not implemented, event 30 is implemented, and filtering on event 30 is supported:

Filter on IMPLEMENTATION DEFINED event 30.

E[30]	Meaning
0b0	Event 30 is ignored.
0b1	Do not record samples that have event 30 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[29], bit [29]
When FEAT_SPEv1p4 is not implemented, event 29 is implemented, and filtering on event 29 is supported:

Filter on IMPLEMENTATION DEFINED event 29.

E[29]	Meaning
0b0	Event 29 is ignored.
0b1	Do not record samples that have event 29 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[28], bit [28]
When FEAT_SPEv1p4 is not implemented, event 28 is implemented, and filtering on event 28 is supported:

Filter on IMPLEMENTATION DEFINED event 28.

E[28]	Meaning
0b0	Event 28 is ignored.
0b1	Do not record samples that have event 28 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[27], bit [27]
When FEAT_SPEv1p4 is not implemented, event 27 is implemented, and filtering on event 27 is supported:

Filter on IMPLEMENTATION DEFINED event 27.

E[27]	Meaning
0b0	Event 27 is ignored.
0b1	Do not record samples that have event 27 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[26], bit [26]

When FEAT_SPEv1p4 is not implemented, event 26 is implemented, and filtering on event 26 is supported:

Filter on IMPLEMENTATION DEFINED event 26.

E[26]	Meaning
0b0	Event 26 is ignored.
0b1	Do not record samples that have event 26 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[25], bit [25]

When (FEAT_SPE_SME is implemented or FEAT_SPEv1p5 is implemented) and event 25 is implemented:

Filter on SMCU or other shared resource operation event.

E[25]	Meaning
0b0	SMCU or other shared resource operation event is ignored.
0b1	Do not record samples that have the SMCU or other shared resource operation event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPEv1p4 is not implemented, event 25 is implemented, and filtering on event 25 is supported:

Filter on IMPLEMENTATION DEFINED event 25.

E[25]	Meaning
0b0	Event 25 is ignored.
0b1	Do not record samples that have event 25 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[24], bit [24]
When FEAT_SPE_SME is implemented:

Filter on Streaming SVE mode event.

E[24]	Meaning
0b0	Streaming SVE mode event is ignored.
0b1	Do not record samples that have the Streaming SVE mode event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPEv1p4 is not implemented, event 24 is implemented, and filtering on event 24 is supported:

Filter on IMPLEMENTATION DEFINED event 24.

E[24]	Meaning
0b0	Event 24 is ignored.
0b1	Do not record samples that have event 24 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[23], bit [23]
When FEAT_SPEv1p4 is implemented and event 23 is implemented:

Filter on Data snooped event.

E[23]	Meaning
0b0	Data snooped event is ignored.
0b1	Do not record samples that have the Data snooped event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[22], bit [22]
When FEAT_SPEv1p4 is implemented and event 22 is implemented:

Filter on Recently fetched event.

E[22]	Meaning
0b0	Recently fetched event is ignored.
0b1	Do not record samples that have the Recently fetched event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[21], bit [21]
When FEAT_SPEv1p4 is implemented and event 21 is implemented:

Filter on Cache data modified event.

E[21]	Meaning
0b0	Cache data modified event is ignored.
0b1	Do not record samples that have the Cache data modified event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[20], bit [20]
When FEAT_SPEv1p4 is implemented and event 20 is implemented:

Filter on Level 2 data cache miss event.

E[20]	Meaning
0b0	Level 2 data cache miss event is ignored.
0b1	Do not record samples that have the Level 2 data cache miss event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[19], bit [19]
When FEAT_SPEv1p4 is implemented and event 19 is implemented:

Filter on Level 2 data cache access event.

E[19]	Meaning
0b0	Level 2 data cache access event is ignored.
0b1	Do not record samples that have the Level 2 data cache access event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[18], bit [18]
When FEAT_SPEv1p1 is implemented and (FEAT_SVE is implemented or FEAT_SME is implemented):

Filter on Empty predicate event.

E[18]	Meaning
0b0	Empty predicate event is ignored.
0b1	Do not record samples that have the Empty predicate event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[17], bit [17]
When FEAT_SPEv1p1 is implemented and (FEAT_SVE is implemented or FEAT_SME is implemented):

Filter on Partial or empty predicate event.

E[17]	Meaning
0b0	Partial or empty predicate event is ignored.
0b1	Do not record samples that have the Partial or empty predicate event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[16], bit [16]
When FEAT_TME is implemented:

Filter on Transactional event.

E[16]	Meaning
0b0	Transactional event is ignored.
0b1	Do not record samples that have the Transactional event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[15], bit [15]
When event 15 is implemented and filtering on event 15 is supported:

Filter on IMPLEMENTATION DEFINED event 15.

E[15]	Meaning
0b0	Event 15 is ignored.
0b1	Do not record samples that have event 15 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[14], bit [14]
When event 14 is implemented and filtering on event 14 is supported:

Filter on IMPLEMENTATION DEFINED event 14.

E[14]	Meaning
0b0	Event 14 is ignored.
0b1	Do not record samples that have event 14 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[13], bit [13]

When event 13 is implemented and filtering on event 13 is supported:

Filter on IMPLEMENTATION DEFINED event 13.

E[13]	Meaning
0b0	Event 13 is ignored.
0b1	Do not record samples that have event 13 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[12], bit [12]

When event 12 is implemented and filtering on event 12 is supported:

Filter on IMPLEMENTATION DEFINED event 12.

E[12]	Meaning
0b0	Event 12 is ignored.
0b1	Do not record samples that have event 12 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[11], bit [11]

When FEAT_SPEv1p1 is implemented:

Filter on Misalignment event.

E[11]	Meaning
0b0	Misalignment event is ignored.
0b1	Do not record samples that have the Misalignment event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[10], bit [10]

When (FEAT_SPEv1p4 is implemented or filtering on event 10 is optionally supported) and event 10 is implemented:

Filter on Remote access event.

E[10]	Meaning
0b0	Remote access event is ignored.
0b1	Do not record samples that have the Remote access event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[9], bit [9]

When (FEAT_SPEv1p4 is implemented or filtering on event 9 is optionally supported) and event 9 is implemented:

Filter on Last Level cache miss event.

E[9]	Meaning
0b0	Last Level cache miss event is ignored.
0b1	Do not record samples that have the Last Level cache miss event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[8], bit [8]

When (FEAT_SPEv1p4 is implemented or filtering on event 8 is optionally supported) and event 8 is implemented:

Filter on Last Level cache access event.

E[8]	Meaning
0b0	Last Level cache access event is ignored.
0b1	Do not record samples that have the Last Level cache access event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[7], bit [7]

Filter on Mispredicted event.

E[7]	Meaning
0b0	Mispredicted event is ignored.
0b1	Do not record samples that have the Mispredicted event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[6], bit [6]

When FEAT_SPEv1p2 is implemented:

Filter on Not taken event.

E[6]	Meaning
0b0	Not taken event is ignored.
0b1	Do not record samples that have the Not taken event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[5], bit [5]

Filter on TLB walk event.

E[5]	Meaning
0b0	TLB walk event is ignored.
0b1	Do not record samples that have the TLB walk event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[4], bit [4]

When FEAT_SPEv1p4 is implemented or filtering on event 4 is optionally supported:

Filter on TLB access event.

E[4]	Meaning
0b0	TLB access event is ignored.
0b1	Do not record samples that have the TLB access event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[3], bit [3]

Filter on Level 1 data cache refill or miss event.

E[3]	Meaning
0b0	Level 1 data cache refill or miss event is ignored.
0b1	Do not record samples that have the Level 1 data cache refill or miss event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[2], bit [2]

When FEAT_SPEv1p4 is implemented or filtering on event 2 is optionally supported:

Filter on Level 1 data cache access event.

E[2]	Meaning
0b0	Level 1 data cache access event is ignored.
0b1	Do not record samples that have the Level 1 data cache access event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[1], bit [1]

When the PE supports sampling of speculative instructions:

Filter on Architecturally retired event.

E[1]	Meaning
0b0	Architecturally retired event is ignored.
0b1	Do not record samples that have the Architecturally retired event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

If the PE does not support the sampling of speculative instructions, or always discards the sample record for speculative instructions, this bit reads as an UNKNOWN value and the PE ignores its value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, UNKNOWN.

Bit [0]

Reserved, RAZ/WI.

Accessing PMSEVFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSEVFR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b101

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMSEVFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x830];
    else
        X[t, 64] = PMSEVFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSEVFR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSEVFR_EL1;
```

```
MSR PMSEVFR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b101

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMSEVFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x830] = X[t, 64];
    else
        PMSEVFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSEVFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSEVFR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSFCR_EL1, Sampling Filter Control Register

The PMSFCR_EL1 characteristics are:

Purpose

Controls sample filtering. The filter is the logical AND of the filters controlled by FDS, FnE, FL, FT, and FE bits. For example, if PMSFCR_EL1.FE is 1 and PMSFCR_EL1.FT is 1, then only samples including the selected types and the selected events will be recorded.

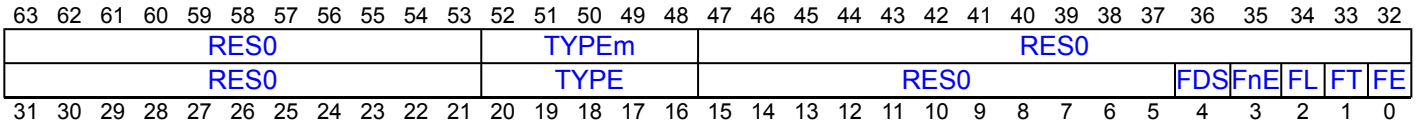
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSFCR_EL1 are UNDEFINED.

Attributes

PMSFCR_EL1 is a 64-bit register.

Field descriptions

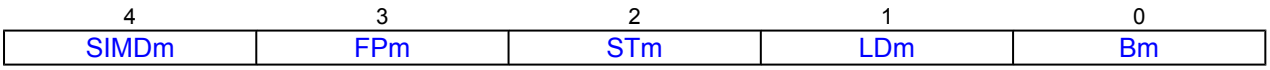


Bits [63:53]

Reserved, RES0.

TYPEm, bits [52:48]

TYPEm encoding when FEAT_SPE_EFT is implemented



SIMDm, bit [4]

SIMD filter mask.

SIMDm	Meaning
0b0	PMSFCR_EL1.SIMD controls whether SIMD operations are recorded as part of a Boolean-OR filter with other masked operation type filter controls.
0b1	PMSFCR_EL1.SIMD controls whether SIMD operations are recorded as part of a Boolean-AND filter with other unmasked operation type filter controls.

This field is ignored by the PE when PMSFCR_EL1.FT is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FPm, bit [3]

Floating-point filter mask.

FPm	Meaning
0b0	PMSFCR_EL1.FP controls whether floating-point operations are recorded as part of a Boolean-OR filter with other masked operation type filter controls.
0b1	PMSFCR_EL1.FP controls whether floating-point operations are recorded as part of a Boolean-AND filter with other unmasked operation type filter controls.

This field is ignored by the PE when PMSFCR_EL1.FT is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

STm, bit [2]

Store filter mask.

STm	Meaning
0b0	PMSFCR_EL1.ST controls whether store operations are recorded as part of a Boolean-OR filter with other masked operation type filter controls.
0b1	PMSFCR_EL1.ST controls whether store operations are recorded as part of a Boolean-AND filter with other unmasked operation type filter controls.

This field is ignored by the PE when PMSFCR_EL1.FT is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LDm, bit [1]

Load filter mask.

LDm	Meaning
0b0	PMSFCR_EL1.LD controls whether load operations are recorded as part of a Boolean-OR filter with other masked operation type filter controls.
0b1	PMSFCR_EL1.LD controls whether load operations are recorded as part of a Boolean-AND filter with other unmasked operation type filter controls.

This field is ignored by the PE when PMSFCR_EL1.FT is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bm, bit [0]

Branch filter mask.

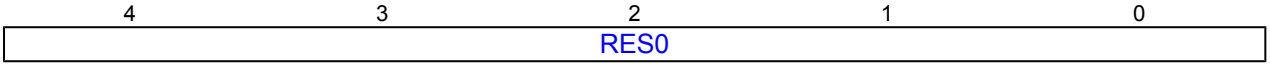
Bm	Meaning
0b0	PMSFCR_EL1.B controls whether branch operations are recorded as part of a Boolean-OR filter with other masked operation type filter controls.
0b1	PMSFCR_EL1.B controls whether branch operations are recorded as part of a Boolean-AND filter with other unmasked operation type filter controls.

This field is ignored by the PE when PMSFCR_EL1.FT is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TYPEm encoding when FEAT_SPE_EFT is not implemented



Bits [4:0]

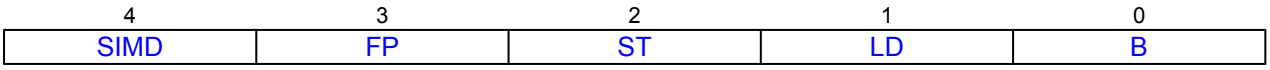
Reserved, RES0.

Bits [47:21]

Reserved, RES0.

TYPE, bits [20:16]

TYPE encoding



SIMD, bit [4]

When FEAT_SPE_EFT is implemented:

SIMD filter enable.

SIMD	Meaning
0b0	If PMSFCR_EL1.SIMDm is 1, then record only operations that are not SIMD operations. Otherwise, do not record SIMD operations, unless allowed by another operation type filter.
0b1	If PMSFCR_EL1.SIMDm is 1, then record only operations that are SIMD operations. Otherwise, record all SIMD operations.

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT is 0.
- PMSFCR_EL1.SIMDm is 0 and the values of the PMSCR_EL1.{SIMD, FP, ST, LD, B} bits for which the corresponding PMSCR_EL1.{SIMDm, FPm, STm, LDm, Bm} bit is zero are all zero.

For filtering purposes, SIMD operations means all Advanced SIMD, SVE, and SME SIMD operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FP, bit [3]

When FEAT_SPE_EFT is implemented:

Floating-point filter enable.

FP	Meaning
0b0	If PMSFCR_EL1.FPm is 1, then record only operations that are not floating-point operations. Otherwise, do not record floating-point operations, unless allowed by another operation type filter.
0b1	If PMSFCR_EL1.FPm is 1, then record only operations that are floating-point operations. Otherwise, record all floating-point operations.

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT is 0.
- PMSFCR_EL1.FPm is 0 and the values of the PMSCR_EL1.{SIMD, FP, ST, LD, B} bits for which the corresponding PMSCR_EL1.{SIMDm, FPm, STm, LDm, Bm} bit is zero are all zero.

For filtering purposes, floating-point operations means all scalar, Advanced SIMD, SVE, and SME floating-point operations, as defined by the FP_SPEC event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ST, bit [2]

Store filter enable.

ST	Meaning
0b0	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.STm is 1, then record only operations that are not store operations. Otherwise, do not record store operations, unless allowed by another operation type filter.
0b1	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.STm is 1, then record only operations that are store operations. Otherwise, record all store operations.

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT is 0.
- FEAT_SPE_EFT is implemented, PMSFCR_EL1.STm is 0, and the values of the PMSCR_EL1.{SIMD, FP, ST, LD, B} bits for which the corresponding PMSCR_EL1.{SIMDm, FPm, STm, LDm, Bm} bit is zero are all zero.

For filtering purposes, store operations includes vector stores and all atomic operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LD, bit [1]

Load filter enable.

LD	Meaning
0b0	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.LDm is 1, then record only operations that are not load operations. Otherwise, do not record load operations, unless allowed by another operation type filter.
0b1	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.LDm is 1, then record only operations that are load operations. Otherwise, record all load operations.

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT is 0.
- FEAT_SPE_EFT is implemented, PMSFCR_EL1.LDm is 0, and the values of the PMSCR_EL1.{SIMD, FP, ST, LD, B} bits for which the corresponding PMSCR_EL1.{SIMDm, FPm, STm, LDm, Bm} bit is zero are all zero.

For filtering purposes, load operations includes vector loads and atomic operations that return a value to the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

B, bit [0]

Branch filter enable.

B	Meaning
0b0	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.Bm is 1, then record only operations that are not branch operations. Otherwise, do not record branch operations.
0b1	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.Bm is 1, then record only operations that are branch operations. Otherwise, record all branch operations.

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT is 0.
- FEAT_SPE_EFT is implemented, PMSFCR_EL1.Bm is 0, and the values of the PMSCR_EL1.{SIMD, FP, ST, LD, B} bits for which the corresponding PMSCR_EL1.{SIMDm, FPm, STm, LDm, Bm} bit is zero are all zero.

For filtering purposes, branch operations includes exception returns.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:5]

Reserved, RES0.

FDS, bit [4]

When FEAT_SPE_FDS is implemented:

Filter by Data Source.

FDS	Meaning
0b0	Data Source filtering disabled.
0b1	Data Source filtering enabled. Samples of load instructions reporting a Data Source not selected by PMSDSFR_EL1 will not be recorded.

If PMSFCR_EL1.FDS is 1 and PMSDSFR_EL1 is zero, then no load operations with a Data Source will be recorded.

Load operations without a Data Source and other sampled operations are unaffected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnE, bit [3]

When FEAT_SPE_FnE is implemented:

Filter by event, inverted.

FnE	Meaning
0b0	Inverted event filtering disabled.
0b1	Inverted event filtering enabled. Samples including the events selected by PMSNEVFR_EL1 will not be recorded.

If any of the following are true, then it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FnE is 0:

- PMSFCR_EL1.FnE is 1 and PMSNEVFR_EL1 is zero.
- PMSFCR_EL1.FnE is 1, PMSFCR_EL1.FE is 1, and there exists a value x such that PMSEVFR_EL1.E[x] is 1 and PMSNEVFR_EL1.E[x] is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FL, bit [2]

Filter by latency.

FL	Meaning
0b0	Latency filtering disabled.
0b1	Latency filtering enabled. Samples with a total latency less than PMSLATFR_EL1.MINLAT will not be recorded.

If this field is 1 and PMSLATFR_EL1.MINLAT is zero, then it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FL is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FT, bit [1]

Filter by type. The filter is controlled by the TYPE and, if FEAT_SPE_EFT is implemented, TYPEm fields.

FT	Meaning
0b0	Type filtering disabled
0b1	Type filtering enabled. Samples not one of the selected types will not be recorded.

Type filtering filters according to the type of the sampled operation.

If FEAT_SPE_EFT is not implemented, this field is 1, and the PMSFCR_EL1.{ST, LD, B} bits are all zero, then it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FT is 0.

For more information, see Filtering by operation type.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FE, bit [0]

Filter by event.

FE	Meaning
0b0	Event filtering disabled.
0b1	Event filtering enabled. Samples not including the events selected by PMSEVFR_EL1 will not be recorded.

If any of the following are true, then it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FE is 0:

- PMSFCR_EL1.FE is 1 and PMSEVFR_EL1 is zero.
- FEAT_SPE_FnE is implemented, PMSFCR_EL1.FnE is 1, PMSFCR_EL1.FE is 1, and there exists a value x such that PMSEVFR_EL1.E[x] is 1 and PMSNEVFR_EL1.E[x] is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSFCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSFCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b100

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMSFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSFCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSFCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSFCR_EL1;
```

```
MSR PMSFCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b100

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMSFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSFCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSICR_EL1, Sampling Interval Counter Register

The PMSICR_EL1 characteristics are:

Purpose

Software must write zero to PMSICR_EL1 before enabling sample profiling for a sampling session. Software must then treat PMSICR_EL1 as an opaque, 64-bit, read/write register used for context switches only.

Configuration

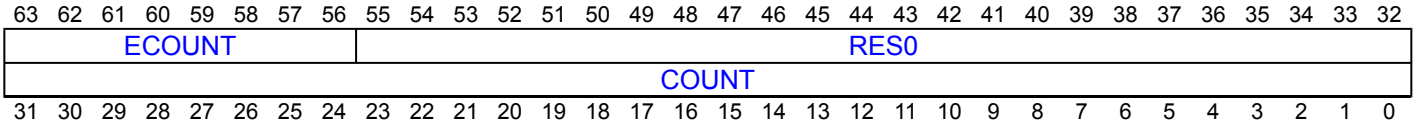
This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSICR_EL1 are UNDEFINED.

The value of PMSICR_EL1 does not change whilst profiling is disabled.

Attributes

PMSICR_EL1 is a 64-bit register.

Field descriptions



ECOUNT, bits [63:56]

When PMSIDR_EL1.ERnd == 1:

Secondary sample interval counter. Provides the secondary counter used after the primary counter reaches zero.

While the secondary counter is nonzero and profiling is enabled, the secondary counter decrements by 1 for each member of the sample population. The primary counter also continues to decrement since it is also nonzero. When the secondary counter reaches zero, a member of the sampling population is selected for sampling.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [55:32]

Reserved, RES0.

COUNT, bits [31:0]

Primary sample interval counter. Provides the primary counter used for sampling.

When the PE moves from a state or Exception level where profiling is disabled to a state or Exception level where profiling is enabled, if the value of this register is zero, then the primary counter is loaded from PMSIRR_EL1.

While the primary counter is nonzero and sampling is enabled, the primary counter decrements by 1 for each member of the sample population.

The sample interval counter counts either a number of operations or a number of instructions, depending on the IMPLEMENTATION DEFINED value of PMSIDR_EL1.ArchInst.

When the counter reaches zero, the behavior depends on the values of PMSIDR_EL1.ERnd and PMSIRR_EL1.RND:

- If PMSIRR_EL1.RND is 1 and PMSIDR_EL1.ERnd is 1, then the secondary counter is set to a random or pseudorandom value in the range `0x00` to `0xFF`.
- Otherwise, a member of the sampling population is selected for sampling.
- The primary counter is loaded from PMSIRR_EL1.

The primary counter is loaded from PMSIRR_EL1 means:

- PMSICR_EL1.COUNT[31:8] is set to the value of PMSIRR_EL1.INTERVAL.
- If PMSIRR_EL1.RND is 1 and PMSIDR_EL1.ERnd is 0, then PMSICR_EL1.COUNT[7:0] is set to a random or pseudorandom value in the range `0x00` to `0xFF`.
- Otherwise, PMSICR_EL1.COUNT[7:0] is set to `0x00`.

For more information, see Initializing the sample interval counters and Behavior of the sample interval counter while profiling is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSICR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSICR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b010

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMSICR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x838];
    else
        X[t, 64] = PMSICR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSICR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSICR_EL1;
```

```
MSR PMSICR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b010

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMSICR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x838] = X[t, 64];
    else
        PMSICR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSICR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSICR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSIDR_EL1, Sampling Profiling ID Register

The PMSIDR_EL1 characteristics are:

Purpose

Describes the Statistical Profiling implementation to software.

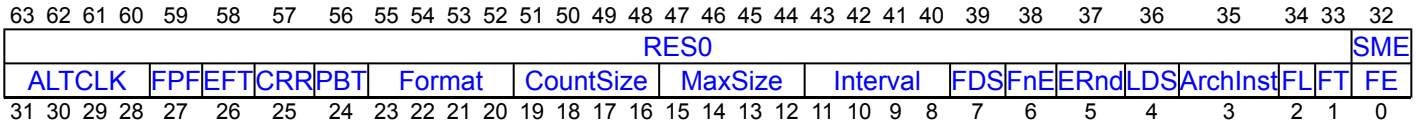
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSIDR_EL1 are UNDEFINED.

Attributes

PMSIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

SME, bit [32]

SPE for SME. Adds support for the Scalable Matrix Extensions to Statistical Profiling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SME	Meaning
0b0	The SME extensions to the Statistical Profiling Extension are not implemented.
0b1	Adds support for Statistical Profiling of the Scalable Matrix Extensions.

FEAT_SPE_SME implements the functionality identified by the value 1.

Access to this field is **RO**.

ALTCLK, bits [31:28]

Alternate clock domain.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ALTCLK	Meaning
0b0000	Alternate clock domain not implemented, or CPU clock domain.
0b0001	The external Streaming Mode Compute Unit provides the alternate clock domain.
0b1111	IMPLEMENTATION DEFINED clock domain.

All other values are reserved.

FEAT_SPE_ALTCLK implements the functionality identified by a nonzero value.

Access to this field is **RO**.

FPF, bit [27]

Floating-point flag.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPF	Meaning
0b0	Operation Type packets for scalar and Advanced SIMD operations do not contain floating-point or Advanced SIMD information.
0b1	Operation Type packets for scalar and Advanced SIMD operations contain floating-point or Advanced SIMD information.

FEAT_SPE_FPF implements the functionality identified by the value 1.

Access to this field is **RO**.

EFT, bit [26]

Extended filtering by operation type.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EFT	Meaning
0b0	PMSFCR_EL1.{SIMDm, FPm, STm, Bm, LDm, SIMD, FP} are RES0.
0b1	PMSFCR_EL1.{SIMDm, FPm, STm, Bm, LDm, SIMD, FP} are implemented.

FEAT_SPE_EFT implements the functionality identified by the value 1.

Access to this field is **RO**.

CRR, bit [25]

Call Return branch records.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CRR	Meaning
0b0	Operation Type packets for branches do not contain Call Return information.
0b1	Operation Type packets for branches contain Call Return information.

FEAT_SPE_CRR implements the functionality identified by the value 1.

Access to this field is **RO**.

PBT, bit [24]

Previous branch target Address packet.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PBT	Meaning
0b0	Previous branch target Address packet not supported.
0b1	Previous branch target Address packet support implemented.

FEAT_SPE_PBT implements the OPTIONAL functionality identified by the value 1.

Access to this field is **RO**.

Format, bits [23:20]

Defines the format of the sample records.

Format	Meaning
0b0000	Format 0.

All other values are reserved.

Access to this field is **RO**.

CountSize, bits [19:16]

Defines the size of the counters.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CountSize	Meaning
0b0010	12-bit saturating counters.
0b0011	16-bit saturating counters.

All other values are reserved.

Access to this field is **RO**.

MaxSize, bits [15:12]

Defines the largest size for a single record, rounded up to a power-of-two. If this is the same as the minimum alignment (PMBIDR_EL1.Align), then each record is exactly this size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MaxSize	Meaning
0b0100	16 bytes.
0b0101	32 bytes.
0b0110	64 bytes.
0b0111	128 bytes.
0b1000	256 bytes.
0b1001	512 bytes.
0b1010	1KB.
0b1011	2KB.

All other values are reserved.

The values 0b0100 and 0b0101 are not permitted for an implementation.

Access to this field is **RO**.

Interval, bits [11:8]

Minimum sampling interval. This provides guidance from the implementer to the smallest sampling interval.

The interval is defined as a number of either operations or instructions, depending on the IMPLEMENTATION DEFINED value of PMSIDR_EL1.ArchInst.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Interval	Meaning
0b0000	256 operations or instructions, or no minimum specified.
0b0010	512 operations or instructions.
0b0011	768 operations or instructions.
0b0100	1,024 operations or instructions.
0b0101	1,536 operations or instructions.
0b0110	2,048 operations or instructions.
0b0111	3,072 operations or instructions.
0b1000	4,096 operations or instructions.

All other values are reserved.

If this field reads as a nonzero value, then setting the sample interval to a value less than the indicated value is likely to cause a statistically significant number of sample collisions. For example, the smallest interval might indicate the typical size of the speculation window for the implementation. That is, the number of operations or instructions that will be executing in parallel, when the processor implementation is not heavily loaded.

Setting the sample interval to a value greater than or equal to the indicated value does not guarantee a statistically insignificant number of sample collisions, as this is typically dependent on the program being profiled. For example, if the program has a large number of long latency loads due to cache misses, then this might cause more collisions when these operations or instructions are sampled.

For more information, see 'Sample collisions'.

Access to this field is **RO**.

FDS, bit [7]

When FEAT_SPEv1p4 is implemented:

Filtering by data source.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FDS	Meaning
0b0	PMSDSFR_EL1 is not implemented and PMSFCR_EL1.FDS is RES0.
0b1	PMSDSFR_EL1 and PMSFCR_EL1.FDS are implemented.

FEAT_SPE_FDS implements the functionality identified by the value 1.

From Armv8.9, if FEAT_SPE_LDS is implemented, the value 0 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

FnE, bit [6]

When FEAT_SPEv1p2 is implemented:

Filtering by events, inverted.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FnE	Meaning
0b0	PMSNEVFR_EL1 is not implemented and PMSFCR_EL1.FnE is RES0.
0b1	PMSNEVFR_EL1 and PMSFCR_EL1.FnE are implemented.

FEAT_SPE_FnE implements the functionality identified by the value 1.

From Armv8.7, if FEAT_SPE is implemented, the value 0 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

ERnd, bit [5]

Defines how the random number generator is used in determining the interval between samples, when enabled by PMSIRR_EL1.RND.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ERnd	Meaning
0b0	The random number is added at the start of the interval, and the sample is taken and a new interval started when the combined interval expires.
0b1	The random number is added and the new interval started after the interval programmed in PMSIRR_EL1.INTERVAL expires, and the sample is taken when the random interval expires.

Access to this field is **RO**.

LDS, bit [4]

Data source indicator for sampled load instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LDS	Meaning
0b0	Loaded data source not implemented.
0b1	Loaded data source implemented.

FEAT_SPE_LDS implements the functionality identified by the value 1.

Access to this field is **RO**.

ArchInst, bit [3]

Architectural instruction profiling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ArchInst	Meaning
0b0	Micro-op sampling implemented.
0b1	Architecture instruction sampling implemented.

Access to this field is **RO**.

FL, bit [2]

Filtering by latency.

Reads as 0b1.

Access to this field is **RO**.

FT, bit [1]

Filtering by operation type.

Reads as 0b1.

Access to this field is **RO**.

FE, bit [0]

Filtering by events.

Reads as 0b1.

Access to this field is **RO**.

Accessing PMSIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b111

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMSIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSIRR_EL1, Sampling Interval Reload Register

The PMSIRR_EL1 characteristics are:

Purpose

Defines the interval between samples.

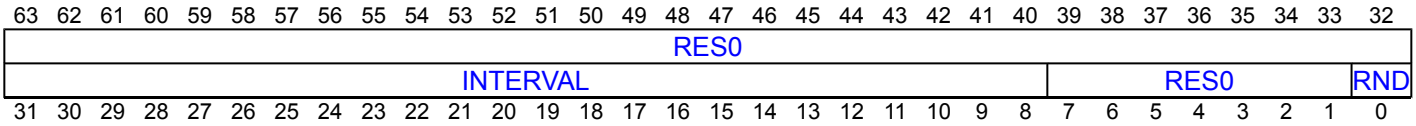
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSIRR_EL1 are UNDEFINED.

Attributes

PMSIRR_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

INTERVAL, bits [31:8]

Bits [31:8] of the PMSICR_EL1 interval counter reload value. Software must set this to a nonzero value.

If this field is zero, then an UNKNOWN sampling interval is used.

If PMSIDR_EL1.Interval is nonzero, then it provides guidance from the implementer to the smallest sampling interval that should be used.

Setting this field to a nonzero value smaller than the indicated value is likely to cause a statistically significant number of sample collisions. See Sample collisions. Setting the sample interval to a value greater than or equal to the indicated value does not guarantee a statistically insignificant number of sample collisions, as this is typically dependent on the program being profiled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:1]

Reserved, RES0.

RND, bit [0]

Controls randomization of the sampling interval.

RND	Meaning
0b0	Disable randomization of sampling interval.
0b1	Add (pseudo-)random jitter to sampling interval.

The random number generator is not architected.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSIRR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSIRR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMSIRR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x840];
    else
        X[t, 64] = PMSIRR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSIRR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSIRR_EL1;
```

```
MSR PMSIRR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMSIRR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x840] = X[t, 64];
    else
        PMSIRR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSIRR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSIRR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSLATFR_EL1, Sampling Latency Filter Register

The PMSLATFR_EL1 characteristics are:

Purpose

Controls sample filtering by latency

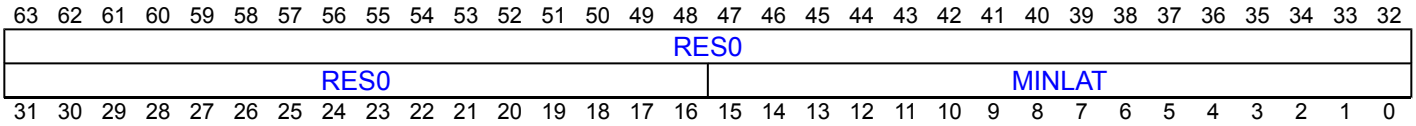
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSLATFR_EL1 are UNDEFINED.

Attributes

PMSLATFR_EL1 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

MINLAT, bits [15:0]

Minimum latency.

When PMSFCR_EL1.FL is 1, defines the value used for filtering by latency. Samples with a Total latency less than PMSLATFR_EL1.MINLAT are not recorded.

If PMSIDR_EL1.CountSize is `0b0010`, PMSLATFR_EL1.MINLAT[15:12] is RES0.

This field is ignored by the PE when PMSFCR_EL1.FL is 0.

For more information, see Filtering by latency.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSLATFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSLATFR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b110

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMSLATFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x848];
    else
        X[t, 64] = PMSLATFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSLATFR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSLATFR_EL1;
```

```
MSR PMSLATFR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b110

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMSLATFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x848] = X[t, 64];
    else
        PMSLATFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSLATFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSLATFR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSNEVFR_EL1, Sampling Inverted Event Filter Register

The PMSNEVFR_EL1 characteristics are:

Purpose

Controls sample filtering by events. The overall inverted filter is the logical OR of these filters. For example, if PMSNEVFR_EL1.E[3] and PMSNEVFR_EL1.E[5] are both set to 1, samples that have either event 3 (Level 1 unified or data cache refill) or event 5 (TLB walk) set to 1 are not recorded.

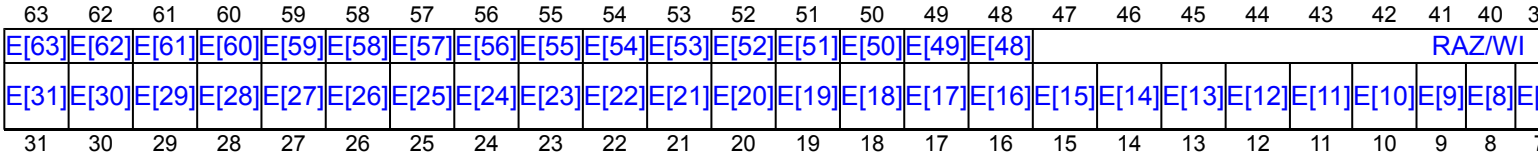
Configuration

This register is present only when FEAT_SPE_FnE is implemented. Otherwise, direct accesses to PMSNEVFR_EL1 are UNDEFINED.

Attributes

PMSNEVFR_EL1 is a 64-bit register.

Field descriptions



E[63], bit [63]

When event 63 is implemented and filtering on event 63 is supported:

Filter on IMPLEMENTATION DEFINED event 63.

E[63]	Meaning
0b0	Event 63 is ignored.
0b1	Do not record samples that have event 63 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[62], bit [62]

When event 62 is implemented and filtering on event 62 is supported:

Filter on IMPLEMENTATION DEFINED event 62.

E[62]	Meaning
0b0	Event 62 is ignored.
0b1	Do not record samples that have event 62 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[61], bit [61]

When event 61 is implemented and filtering on event 61 is supported:

Filter on IMPLEMENTATION DEFINED event 61.

E[61]	Meaning
0b0	Event 61 is ignored.
0b1	Do not record samples that have event 61 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[60], bit [60]

When event 60 is implemented and filtering on event 60 is supported:

Filter on IMPLEMENTATION DEFINED event 60.

E[60]	Meaning
0b0	Event 60 is ignored.
0b1	Do not record samples that have event 60 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[59], bit [59]

When event 59 is implemented and filtering on event 59 is supported:

Filter on IMPLEMENTATION DEFINED event 59.

E[59]	Meaning
0b0	Event 59 is ignored.
0b1	Do not record samples that have event 59 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[58], bit [58]

When event 58 is implemented and filtering on event 58 is supported:

Filter on IMPLEMENTATION DEFINED event 58.

E[58]	Meaning
0b0	Event 58 is ignored.
0b1	Do not record samples that have event 58 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[57], bit [57]

When event 57 is implemented and filtering on event 57 is supported:

Filter on IMPLEMENTATION DEFINED event 57.

E[57]	Meaning
0b0	Event 57 is ignored.
0b1	Do not record samples that have event 57 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[56], bit [56]

When event 56 is implemented and filtering on event 56 is supported:

Filter on IMPLEMENTATION DEFINED event 56.

E[56]	Meaning
0b0	Event 56 is ignored.
0b1	Do not record samples that have event 56 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[55], bit [55]

When event 55 is implemented and filtering on event 55 is supported:

Filter on IMPLEMENTATION DEFINED event 55.

E[55]	Meaning
0b0	Event 55 is ignored.
0b1	Do not record samples that have event 55 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[54], bit [54]

When event 54 is implemented and filtering on event 54 is supported:

Filter on IMPLEMENTATION DEFINED event 54.

E[54]	Meaning
0b0	Event 54 is ignored.
0b1	Do not record samples that have event 54 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[53], bit [53]
When event 53 is implemented and filtering on event 53 is supported:

Filter on IMPLEMENTATION DEFINED event 53.

E[53]	Meaning
0b0	Event 53 is ignored.
0b1	Do not record samples that have event 53 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[52], bit [52]
When event 52 is implemented and filtering on event 52 is supported:

Filter on IMPLEMENTATION DEFINED event 52.

E[52]	Meaning
0b0	Event 52 is ignored.
0b1	Do not record samples that have event 52 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[51], bit [51]
When event 51 is implemented and filtering on event 51 is supported:

Filter on IMPLEMENTATION DEFINED event 51.

E[51]	Meaning
0b0	Event 51 is ignored.
0b1	Do not record samples that have event 51 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[50], bit [50]
When event 50 is implemented and filtering on event 50 is supported:

Filter on IMPLEMENTATION DEFINED event 50.

E[50]	Meaning
0b0	Event 50 is ignored.
0b1	Do not record samples that have event 50 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[49], bit [49]
When event 49 is implemented and filtering on event 49 is supported:

Filter on IMPLEMENTATION DEFINED event 49.

E[49]	Meaning
0b0	Event 49 is ignored.
0b1	Do not record samples that have event 49 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[48], bit [48]

When event 48 is implemented and filtering on event 48 is supported:

Filter on IMPLEMENTATION DEFINED event 48.

E[48]	Meaning
0b0	Event 48 is ignored.
0b1	Do not record samples that have event 48 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bits [47:32]

Reserved, RAZ/WI.

E[31], bit [31]

When FEAT_SPEv1p4 is not implemented, event 31 is implemented, and filtering on event 31 is supported:

Filter on IMPLEMENTATION DEFINED event 31.

E[31]	Meaning
0b0	Event 31 is ignored.
0b1	Do not record samples that have event 31 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[30], bit [30]

When FEAT_SPEv1p4 is not implemented, event 30 is implemented, and filtering on event 30 is supported:

Filter on IMPLEMENTATION DEFINED event 30.

E[30]	Meaning
0b0	Event 30 is ignored.
0b1	Do not record samples that have event 30 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[29], bit [29]
When FEAT_SPEv1p4 is not implemented, event 29 is implemented, and filtering on event 29 is supported:

Filter on IMPLEMENTATION DEFINED event 29.

E[29]	Meaning
0b0	Event 29 is ignored.
0b1	Do not record samples that have event 29 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[28], bit [28]
When FEAT_SPEv1p4 is not implemented, event 28 is implemented, and filtering on event 28 is supported:

Filter on IMPLEMENTATION DEFINED event 28.

E[28]	Meaning
0b0	Event 28 is ignored.
0b1	Do not record samples that have event 28 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[27], bit [27]
When FEAT_SPEv1p4 is not implemented, event 27 is implemented, and filtering on event 27 is supported:

Filter on IMPLEMENTATION DEFINED event 27.

E[27]	Meaning
0b0	Event 27 is ignored.
0b1	Do not record samples that have event 27 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[26], bit [26]

When FEAT_SPEv1p4 is not implemented, event 26 is implemented, and filtering on event 26 is supported:

Filter on IMPLEMENTATION DEFINED event 26.

E[26]	Meaning
0b0	Event 26 is ignored.
0b1	Do not record samples that have event 26 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[25], bit [25]

When (FEAT_SPE_SME is implemented or FEAT_SPEv1p5 is implemented) and event 25 is implemented:

Filter on Not SMCU or other shared resource operation event.

E[25]	Meaning
0b0	SMCU or other shared resource operation event is ignored.
0b1	Do not record samples that have the SMCU or other shared resource operation event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPEv1p4 is not implemented, event 25 is implemented, and filtering on event 25 is supported:

Filter on IMPLEMENTATION DEFINED event 25.

E[25]	Meaning
0b0	Event 25 is ignored.
0b1	Do not record samples that have event 25 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[24], bit [24]
When FEAT_SPE_SME is implemented:

Filter on Non-streaming SVE mode event.

E[24]	Meaning
0b0	Streaming SVE mode event is ignored.
0b1	Do not record samples that have the Streaming SVE mode event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPEv1p4 is not implemented, event 24 is implemented, and filtering on event 24 is supported:

Filter on IMPLEMENTATION DEFINED event 24.

E[24]	Meaning
0b0	Event 24 is ignored.
0b1	Do not record samples that have event 24 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[23], bit [23]
When FEAT_SPEv1p4 is implemented and event 23 is implemented:

Filter on Data not snooped event.

E[23]	Meaning
0b0	Data snooped event is ignored.
0b1	Do not record samples that have the Data snooped event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[22], bit [22]
When FEAT_SPEv1p4 is implemented and event 22 is implemented:

Filter on Not recently fetched event.

E[22]	Meaning
0b0	Recently fetched event is ignored.
0b1	Do not record samples that have the Recently fetched event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[21], bit [21]
When FEAT_SPEv1p4 is implemented and event 21 is implemented:

Filter on Cache data not modified event.

E[21]	Meaning
0b0	Cache data modified event is ignored.
0b1	Do not record samples that have the Cache data modified event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[20], bit [20]
When FEAT_SPEv1p4 is implemented and event 20 is implemented:

Filter on Level 2 data cache hit event.

E[20]	Meaning
0b0	Level 2 data cache miss event is ignored.
0b1	Do not record samples that have the Level 2 data cache miss event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[19], bit [19]
When FEAT_SPEv1p4 is implemented and event 19 is implemented:

Filter on No level 2 data cache access event.

E[19]	Meaning
0b0	Level 2 data cache access event is ignored.
0b1	Do not record samples that have the Level 2 data cache access event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[18], bit [18]
When FEAT_SPEv1p1 is implemented and (FEAT_SVE is implemented or FEAT_SME is implemented):

Filter on Not empty predicate event.

E[18]	Meaning
0b0	Empty predicate event is ignored.
0b1	Do not record samples that have the Empty predicate event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[17], bit [17]
When FEAT_SPEv1p1 is implemented and (FEAT_SVE is implemented or FEAT_SME is implemented):

Filter on Not partial predicate event.

E[17]	Meaning
0b0	Partial or empty predicate event is ignored.
0b1	Do not record samples that have the Partial or empty predicate event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[16], bit [16]
When FEAT_TME is implemented:

Filter on Not transactional event.

E[16]	Meaning
0b0	Transactional event is ignored.
0b1	Do not record samples that have the Transactional event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[15], bit [15]
When event 15 is implemented and filtering on event 15 is supported:

Filter on IMPLEMENTATION DEFINED event 15.

E[15]	Meaning
0b0	Event 15 is ignored.
0b1	Do not record samples that have event 15 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[14], bit [14]
When event 14 is implemented and filtering on event 14 is supported:

Filter on IMPLEMENTATION DEFINED event 14.

E[14]	Meaning
0b0	Event 14 is ignored.
0b1	Do not record samples that have event 14 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[13], bit [13]
When event 13 is implemented and filtering on event 13 is supported:

Filter on IMPLEMENTATION DEFINED event 13.

E[13]	Meaning
0b0	Event 13 is ignored.
0b1	Do not record samples that have event 13 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[12], bit [12]
When event 12 is implemented and filtering on event 12 is supported:

Filter on IMPLEMENTATION DEFINED event 12.

E[12]	Meaning
0b0	Event 12 is ignored.
0b1	Do not record samples that have event 12 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[11], bit [11]

When FEAT_SPEv1p1 is implemented:

Filter on Aligned event.

E[11]	Meaning
0b0	Misalignment event is ignored.
0b1	Do not record samples that have the Misalignment event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[10], bit [10]

When (FEAT_SPEv1p4 is implemented or filtering on event 10 is optionally supported) and event 10 is implemented:

Filter on No remote access event.

E[10]	Meaning
0b0	Remote access event is ignored.
0b1	Do not record samples that have the Remote access event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[9], bit [9]

When (FEAT_SPEv1p4 is implemented or filtering on event 9 is optionally supported) and event 9 is implemented:

Filter on Last Level cache hit event.

E[9]	Meaning
0b0	Last Level cache miss event is ignored.
0b1	Do not record samples that have the Last Level cache miss event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[8], bit [8]

When (FEAT_SPEv1p4 is implemented or filtering on event 8 is optionally supported) and event 8 is implemented:

Filter on No Last Level cache access event.

E[8]	Meaning
0b0	Last Level cache access event is ignored.
0b1	Do not record samples that have the Last Level cache access event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[7], bit [7]

Filter on Correctly predicted event.

E[7]	Meaning
0b0	Mispredicted event is ignored.
0b1	Do not record samples that have the Mispredicted event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[6], bit [6]

When FEAT_SPEv1p2 is implemented:

Filter on Taken event.

E[6]	Meaning
0b0	Not taken event is ignored.
0b1	Do not record samples that have the Not taken event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[5], bit [5]

Filter on TLB hit event.

E[5]	Meaning
0b0	TLB walk event is ignored.
0b1	Do not record samples that have the TLB walk event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[4], bit [4]

When FEAT_SPEv1p4 is implemented or filtering on event 4 is optionally supported:

Filter on No TLB access event.

E[4]	Meaning
0b0	TLB access event is ignored.
0b1	Do not record samples that have the TLB access event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[3], bit [3]

Filter on Level 1 data cache hit event.

E[3]	Meaning
0b0	Level 1 data cache refill or miss event is ignored.
0b1	Do not record samples that have the Level 1 data cache refill or miss event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[2], bit [2]

When FEAT_SPEv1p4 is implemented or filtering on event 2 is optionally supported:

Filter on No Level 1 data cache access event.

E[2]	Meaning
0b0	Level 1 data cache access event is ignored.
0b1	Do not record samples that have the Level 1 data cache access event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[1], bit [1]

When the PE supports sampling of speculative instructions:

Filter on Speculative event.

E[1]	Meaning
0b0	Architecturally retired event is ignored.
0b1	Do not record samples that have the Architecturally retired event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

If the PE does not support the sampling of speculative instructions, or always discards the sample record for speculative instructions, this bit reads as an UNKNOWN value and the PE ignores its value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bit [0]

Reserved, RAZ/WI.

Accessing PMSNEVFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSNEVFR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b001

```
if !IsFeatureImplemented(FEAT_SPE_FnE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.nPMSNEVFR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x850];
    else
        X[t, 64] = PMSNEVFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSNEVFR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSNEVFR_EL1;
```

```
MSR PMSNEVFR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b001

```
if !IsFeatureImplemented(FEAT_SPE_FnE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.nPMSNEVFR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x850] = X[t, 64];
    else
        PMSNEVFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSNEVFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSNEVFR_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSSCR_EL1, Performance Monitors Snapshot Status and Capture Register

The PMSSCR_EL1 characteristics are:

Purpose

Holds status information about the captured counters and provides a mechanism for software to initiate a sample.

Configuration

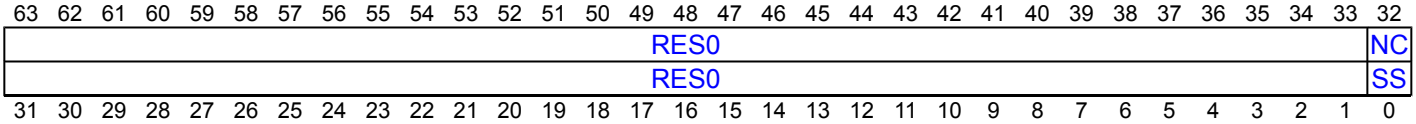
AArch64 System register PMSSCR_EL1 bits [63:0] are architecturally mapped to External register PMSSCR_EL1[63:0].

This register is present only when FEAT_PMUv3_SS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMSSCR_EL1 are UNDEFINED.

Attributes

PMSSCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

NC, bit [32]

No Capture. Indicates whether the PMU counters have been captured.

NC	Meaning
0b0	PMU counters captured.
0b1	PMU counters not captured.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

Bits [31:1]

Reserved, RES0.

SS, bit [0]

Snapshot Capture and Status.

SS	Meaning
0b0	On a read, the Capture event has completed. On a write, request a Capture event.
0b1	On a read, the Capture event has not completed. On a write, request a Capture event.

A write of 0 to this field is ignored.

It is CONSTRAINED UNPREDICTABLE whether a Capture event has completed if this field is modified when the Capture event is ongoing.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- When PMU capture events are disabled, access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Accessing PMSSCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSSCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3_SS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMSSCR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSSCR_EL1;
```

```
MSR PMSSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3_SS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nPMSSCR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSSCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSWINC_EL0, Performance Monitors Software Increment Register

The PMSWINC_EL0 characteristics are:

Purpose

Increments a counter that is configured to count the Software increment event, event `0x00`. For more information, see 'SW_INCR'.

Configuration

AArch64 System register PMSWINC_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMSWINC[31:0].

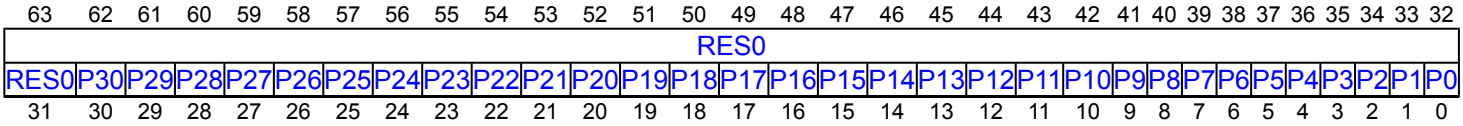
AArch64 System register PMSWINC_EL0 bits [31:0] are architecturally mapped to External register PMSWINC_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented.

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMSWINC_EL0 are UNDEFINED.

Attributes

PMSWINC_EL0 is a 64-bit register.

Field descriptions



Bits [63:31]

Reserved, RES0.

P<m>, bit [m], for m = 30 to 0

Software increment.

P<m>	Meaning
`0b0`	Write is ignored.
`0b1`	Increment PMEVCNTR<m>_EL0, if PMEVCNTR<m>_EL0 is configured to count software increment events.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,SW] == 0b10.
  - PMUACR_EL1.P<m> == 0.
- Otherwise, access to this field is **WO/RAZ**.

Accessing PMSWINC_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MSR PMSWINC_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,SW,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<SW,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSWINC_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSWINC_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMSWINC_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSWINC_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSWINC_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSWINC_EL0 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMUACR_EL1, Performance Monitors User Access Control Register

The PMUACR_EL1 characteristics are:

Purpose

Enables or disables EL0 access to specfic Performance Monitors.

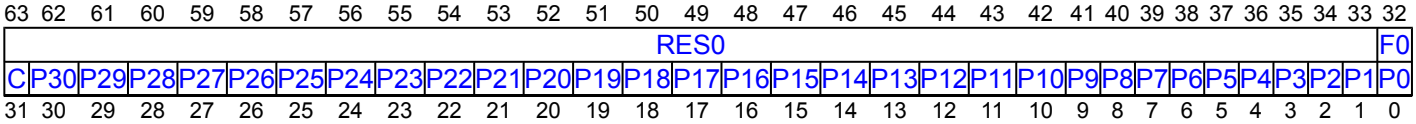
Configuration

This register is present only when FEAT_PMUv3p9 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMUACR_EL1 are UNDEFINED.

Attributes

PMUACR_EL1 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

EL0 accesses to PMICNTR_EL0 enable. With PMUSERENR_EL0.UEN and PMUSERENR_EL0.IR, controls EL0 accesses to PMICNTR_EL0.

F0	Meaning
0b0	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMICNTR_EL0 and associated controls are RAZ/WI, and permitted EL0 writes to PMZR_EL0.F0 are ignored.
0b1	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMICNTR_EL0 and associated controls are either read-only or read/write, depending on the value of PMUSERENR_EL0.IR.

The controls associated with PMICNTR_EL0 that are accessible at EL0 are PMCNTENSET_EL0.F0, PMCNTENCLR_EL0.F0, PMOVSSET_EL0.F0, and PMOVSCLR_EL0.F0.

This field is ignored by the PE when any of the following are true:

- EL1 is using AArch32.
- PMUSERENR_EL0.UEN is 0.
- The access generates an exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

C, bit [31]

EL0 accesses to PMCCNTR_EL0 enable. With PMUSERENR_EL0.EN, PMUSERENR_EL0.UEN, and PMUSERENR_EL0.CR, controls EL0 accesses to PMCCNTR_EL0.

C	Meaning
0b0	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMCCNTR_EL0 and associated controls are RAZ/WI, and permitted EL0 writes to PMZR_EL0.C are ignored.
0b1	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMCCNTR_EL0 and associated controls are either read-only or read/write, depending on the value of PMUSERENR_EL0.CR.

The controls associated with PMCCNTR_EL0 that are accessible at EL0 are PMCNTENSET_EL0.C, PMCNTENCLR_EL0.C, PMOVSSET_EL0.C, and PMOVSCLR_EL0.C.

This field is ignored by the PE when any of the following are true:

- EL1 is using AArch32.
- PMUSERENR_EL0.UEN is 0.
- The access generates an exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<m>, bit [m], for m = 30 to 0

EL0 accesses to PMEVCNTR<m>_EL0 enable. With PMUSERENR_EL0.EN, PMUSERENR_EL0.UEN, and PMUSERENR_EL0.ER, controls EL0 accesses to PMEVCNTR<m>_EL0.

P<m>	Meaning
0b0	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMEVCNTR<m>_EL0 and associated controls are RAZ/WI, and permitted EL0 writes to PMZR_EL0.P<m> are ignored.
0b1	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMEVCNTR<m>_EL0 and associated controls are either read-only or read/write, depending on the value of PMUSERENR_EL0.ER.

The controls associated with PMEVCNTR<m>_EL0 that are accessible at EL0 are PMCNTENSET_EL0.P<m>, PMCNTENCLR_EL0.P<m>, PMOVSSET_EL0.P<m>, and PMOVSCLR_EL0.P<m>.

This field is ignored by the PE when any of the following are true:

- EL1 is using AArch32.
- PMUSERENR_EL0.UEN is 0.
- The access generates an exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RW**.

Accessing PMUACR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMUACR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b100

```
if !(IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nPMUACR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMUACR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMUACR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMUACR_EL1;
```

MSR PMUACR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b100

```
if !(IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nPMUACR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMUACR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMUACR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMUACR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMUSERENR_EL0, Performance Monitors User Enable Register

The PMUSERENR_EL0 characteristics are:

Purpose

Enables or disables EL0 access to the Performance Monitors.

Configuration

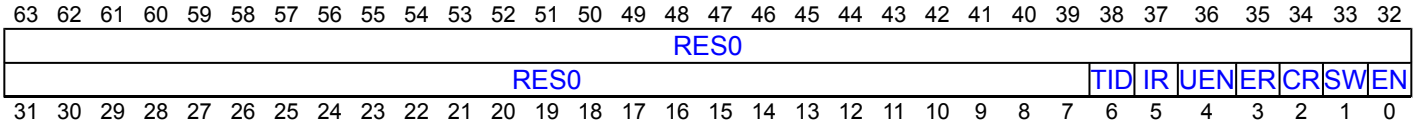
AArch64 System register PMUSERENR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMUSERENR[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMUSERENR_EL0 are UNDEFINED.

Attributes

PMUSERENR_EL0 is a 64-bit register.

Field descriptions



Bits [63:7]

Reserved, RES0.

TID, bit [6]

When FEAT_PMUv3p9 is implemented:

Trap ID registers. Traps EL0 read access to common event identification registers.

TID	Meaning
0b0	Accesses to PMCEID<n>_EL0 and PMCEID<n> are not trapped by this mechanism.
0b1	EL0 read accesses to PMCEID<n>_EL0 and PMCEID<n> are trapped.

In AArch64 state, the register accesses affected by this control are:

- MRS reads of PMCEID0_EL0 and PMCEID1_EL0.

In AArch32 state, the register accesses affected by this control are:

- MRC reads of PMCEID0, PMCEID1, PMCEID2, and PMCEID3.

When trapped, reads generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MRS reads are reported using EC syndrome value 0x18.
- AArch32 MRC reads are reported using EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IR, bit [5]
When FEAT_PMUv3_ICNTR is implemented:

Instruction counter Read-only. When PMUSERENR_EL0.UEN is 1, controls whether EL0 writes to instruction counter are ignored.

IR	Meaning
0b0	EL0 accesses are not affected by this mechanism.
0b1	If the Effective value of PMUSERENR_EL0.UEN is 1, then all of the following apply at EL0: • Writes to PMICNTR_EL0 are ignored. • The controls associated with instruction counter are RAZ/WI. • Writes to PMZR_EL0.F0 are ignored.

The controls associated with PMICNTR_EL0 that are accessible at EL0 are PMCNTENSET_EL0.F0, PMCNTENCLR_EL0.F0, PMOVSSET_EL0.F0, and PMOVSCLR_EL0.F0.

Ignored writes are not trapped and do not generate an exception.

This field is ignored by the PE when any of the following are true:

- PMUSERENR_EL0.UEN is 0.
- The access generates an exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UEN, bit [4]
When FEAT_PMUv3p9 is implemented:

User Enable, with access controlled by PMUACR_EL1. Enables EL0 read/write access to PMU registers, other than PMCR_EL0.

UEN	Meaning
0b0	If FEAT_PMUv3_ICNTR is implemented, then EL0 accesses to PMICFILTR_EL0 and PMICNTR_EL0 are trapped. EL0 accesses to the other specified PMU registers, PMCR_EL0, and PMCR are trapped, unless enabled by PMUSERENR_EL0.{ER,CR,SW,EN}.
0b1	EL0 accesses to the specified PMU registers are enabled, unless trapped by another control. The behavior of permitted accesses is controlled by PMUSERENR_EL0.{ IR, ER,CR,SW} and PMUACR_EL1. EL0 accesses to PMCR_EL0 and PMCR are trapped.

In AArch64 state, the register accesses affected by this control are:

- MRS or MSR accesses to the following registers:
  - PMCCFILTR_EL0, PMCCNTR_EL0, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMOVSCLR_EL0, PMOVSSET_EL0, PMSELR_EL0, PMXEVCNTR_EL0, and PMXEVTYPER_EL0.
  - If FEAT_PMUv3_ICNTR is implemented, PMICFILTR_EL0 and PMICNTR_EL0.
- MRS reads of PMCEID0_EL0 and PMCEID1_EL0.
- MSR writes to PMSWINC_EL0 and PMZR_EL0.

In AArch32 state, the register accesses affected by this control are:

- MRC or MCR accesses to PMCCFILTR, PMCCNTR, PMCNTENCLR, PMCNTENSET, PMEVCNTR<n>, PMEVTYPER<n>, PMOVSR, PMOVSSET, PMSELR, PMXEVCNTR, and PMXEVTYPER.
- MRC reads of PMCEID0, PMCEID1, PMCEID2, and PMCEID3.
- MCR writes to PMSWINC.
- MRRC or MCRR accesses to PMCCNTR.

When trapped, reads and writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MRS and MSR accesses are reported using EC syndrome value 0x18.
- AArch32 MRC and MCR accesses are reported using EC syndrome value 0x03.
- AArch32 MRRC and MCRR accesses are reported using EC syndrome value 0x04.

This field is ignored by the PE and treated as zero when EL1 is using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ER, bit [3]
When FEAT_PMUv3p9 is implemented:

Event counters Read enable or Read-only.

When PMUSERENR_EL0.{UEN,EN} is {0,0}, PMUSERENR_EL0.ER enables EL0 reads of the event counters and EL0 reads and writes of the select register.

When PMUSERENR_EL0.UEN is 1, EL0 reads of the event counters and EL0 writes to PMZR_EL0 are enabled by PMUSERENR_EL0.UEN, unless trapped by another control, and PMUSERENR_EL0.ER controls the behavior of EL0 writes to the event counters and PMZR_EL0.

ER	Meaning
0b0	When PMUSERENR_EL0.UEN == 0, EL0 reads of the event counters and EL0 reads and writes of the select register are disabled, unless enabled by PMUSERENR_EL0.EN. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes are not affected by this mechanism.
0b1	When PMUSERENR_EL0.UEN == 0, EL0 reads of the event counters and EL0 reads and writes of the select register are enabled, unless trapped by another control. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes to the event counters and PMZR_EL0.P[30:0] are ignored.

In AArch64 state, the register accesses affected by this control are:

- When PMUSERENR_EL0.{UEN,EN} is {0,0}:
  - MRS reads of PMEVCNTR<n>_EL0 and PMXEVCNTR_EL0.
  - MRS and MSR accesses to PMSELR_EL0.
- When PMUSERENR_EL0.UEN is 1, MSR writes to PMZR_EL0.P<n>, PMEVCNTR<n>_EL0, and PMXEVCNTR_EL0.

In AArch32 state, the register accesses affected by this control are:

- When PMUSERENR_EL0.{UEN,EN} is {0,0}:
  - MRC reads of PMEVCNTR<n> and PMXEVCNTR.
  - MRC and MCR accesses to PMSELR.
- When PMUSERENR_EL0.UEN is 1, MCR writes to PMEVCNTR<n> and PMXEVCNTR.

When disabled, reads and writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 `MRS` and `MSR` accesses are reported using EC syndrome value `0x18`.
- AArch32 `MRC` and `MCR` accesses are reported using EC syndrome value `0x03`.

Ignored writes are not trapped and do not generate an exception.

This field is ignored by the PE when PMUSERENR_EL0.{UEN,EN} == {0,1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Event counters Read enable.

When PMUSERENR_EL0.EN is 0, PMUSERENR_EL0.ER enables EL0 reads of the event counters and EL0 reads and writes of the select register.

ER	Meaning
`0b0`	EL0 reads of the event counters and EL0 reads and writes of the select register are disabled, unless enabled by PMUSERENR_EL0.EN.
`0b1`	EL0 reads of the event counters and EL0 reads and writes of the select register are enabled, unless trapped by another control.

In AArch64 state, the register accesses affected by this control are:

- `MRS` reads of PMEVCNTR<n>_EL0 and PMXEVCNTR_EL0.
- `MRS` and `MSR` accesses to PMSELR_EL0.

In AArch32 state, the register accesses affected by this control are:

- `MRC` reads of PMEVCNTR<n> and PMXEVCNTR.
- `MRC` and `MCR` accesses to PMSELR.

When disabled, reads and writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 `MRS` and `MSR` accesses are reported using EC syndrome value `0x18`.
- AArch32 `MRC` and `MCR` accesses are reported using EC syndrome value `0x03`.

This field is ignored by the PE when PMUSERENR_EL0.EN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CR, bit [2]

When FEAT_PMUv3p9 is implemented:

Cycle counter Read enable or Read-only.

When PMUSERENR_EL0.{UEN,EN} is {0,0}, PMUSERENR_EL0.CR enables EL0 reads of the cycle counter.

When PMUSERENR_EL0.UEN is 1, EL0 reads of the cycle counter and EL0 writes to PMZR_EL0 are enabled by PMUSERENR_EL0.UEN, unless trapped by another control, and PMUSERENR_EL0.CR controls the behavior of EL0 writes to the cycle counter and PMZR_EL0.

CR	Meaning
0b0	When PMUSERENR_EL0.UEN == 0, EL0 reads of the cycle counter are disabled, unless enabled by PMUSERENR_EL0.EN. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes are not affected by this mechanism.
0b1	When PMUSERENR_EL0.UEN == 0, EL0 reads of the cycle counter are enabled, unless trapped by another control. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes to the cycle counter and PMZR_EL0.C are ignored.

In AArch64 state, the register accesses affected by this control are:

- When PMUSERENR_EL0.{UEN,EN} is {0,0}, MRS reads of PMCCNTR_EL0.
- When PMUSERENR_EL0.UEN is 1, MSR writes to PMZR_EL0.C and PMCCNTR_EL0.

In AArch32 state, the register accesses affected by this control are:

- When PMUSERENR_EL0.{UEN,EN} is {0,0}:
  - MRC reads of PMCCNTR.
  - MRRC reads of PMCCNTR.
- When PMUSERENR_EL0.UEN is 1:
  - MCR writes to PMCCNTR.
  - MCRR writes to PMCCNTR.

When disabled, reads generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MRS reads are reported using EC syndrome value 0x18.
- AArch32 MRC reads are reported using EC syndrome value 0x03.
- AArch32 MRRC reads are reported using EC syndrome value 0x04.

Ignored writes are not trapped and do not generate an exception.

This field is ignored by the PE when PMUSERENR_EL0.{UEN,EN} == {0,1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Cycle counter Read enable.

When PMUSERENR_EL0.EN is 0, PMUSERENR_EL0.CR enables EL0 reads of the cycle counter.

CR	Meaning
0b0	EL0 reads of the cycle counter are disabled, unless enabled by PMUSERENR_EL0.EN.
0b1	EL0 reads of the cycle counter are enabled, unless trapped by another control.

In AArch64 state, the register accesses affected by this control are:

- MRS reads of PMCCNTR_EL0.

In AArch32 state, the register accesses affected by this control are:

- MRC reads of PMCCNTR.
- MRRC reads of PMCCNTR.

When disabled, reads generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 `MRS` reads are reported using EC syndrome value `0x18`.
- AArch32 `MRC` reads are reported using EC syndrome value `0x03`.
- AArch32 `MRRC` reads are reported using EC syndrome value `0x04`.

This field is ignored by the PE when PMUSERENR_EL0.EN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SW, bit [1]

When FEAT_PMUv3p9 is implemented:

Software increment register Write enable.

When PMUSERENR_EL0.UEN is 0, PMUSERENR_EL0.SW enables EL0 writes to the Software increment register.

When PMUSERENR_EL0.UEN is 1, EL0 writes to the Software increment register are enabled by PMUSERENR_EL0.UEN, unless trapped by another control, and PMUSERENR_EL0.SW controls the behavior of EL0 writes to the Software increment register.

SW	Meaning
`0b0`	When PMUSERENR_EL0.UEN == 0, EL0 writes to the Software increment register are disabled, unless enabled by PMUSERENR_EL0.EN. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes are not affected by this mechanism.
`0b1`	When PMUSERENR_EL0.UEN == 0, EL0 writes to the Software increment register are enabled, unless trapped by another control. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes to the Software increment register ignore the value of PMUACR_EL1.

In AArch64 state, the register accesses affected by this control are:

- `MSR` writes to PMSWINC_EL0.

In AArch32 state, the register accesses affected by this control are:

- `MCR` writes to PMSWINC.

When disabled, writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 `MSR` writes are reported using EC syndrome value `0x18`.
- AArch32 `MCR` writes are reported using EC syndrome value `0x03`.

This field is ignored by the PE when PMUSERENR_EL0.{UEN,EN} == {0,1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Software increment register Write enable.

When PMUSERENR_EL0.EN is 0, PMUSERENR_EL0.SW enables EL0 writes to the Software increment register.

SW	Meaning
`0b0`	EL0 writes to the Software increment register are disabled, unless enabled by PMUSERENR_EL0.EN.
`0b1`	EL0 writes to the Software increment register are enabled, unless trapped by another control.

In AArch64 state, the register accesses affected by this control are:

- `MSR` writes to PMSWINC_EL0.

In AArch32 state, the register accesses affected by this control are:

- MCR writes to PMSWINC.

When disabled, writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MSR writes are reported using EC syndrome value 0x18.
- AArch32 MCR writes are reported using EC syndrome value 0x03.

This field is ignored by the PE when PMUSERENR_EL0.EN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EN, bit [0]

Enable.

Enables EL0 read/write access to PMU registers, other than the instruction counter.

EN	Meaning
0b0	EL0 accesses to the specified PMU System registers are trapped, unless enabled by PMUSERENR_EL0.{UEN,ER,CR,SW}.
0b1	EL0 accesses to the specified PMU System registers are enabled, unless trapped by another control.

In AArch64 state, the register accesses affected by this control are:

- MRS or MSR accesses to PMCCFILTR_EL0, PMCCNTR_EL0, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMOVSCLR_EL0, PMOVSSET_EL0, PMSELR_EL0, PMXEVCNTR_EL0, and PMXEVTYPER_EL0.
- MRS reads of PMCEID0_EL0 and PMCEID1_EL0.
- MSR writes to the following registers:
  - PMSWINC_EL0.
  - If FEAT_PMUv3p9 is implemented, PMZR_EL0.

Note

When FEAT_PMUv3_ICNTR is implemented, this field does not affect MRS and MSR accesses to PMICNTR_EL0 and PMICFILTR_EL0.

In AArch32 state, the register accesses affected by this control are:

- MRC or MCR accesses to PMCCFILTR, PMCCNTR, PMCNTENCLR, PMCNTENSET, PMCR, PMEVCNTR<n>, PMEVTYPER<n>, PMOVSR, PMOVSSET, PMSELR, PMXEVCNTR, and PMXEVTYPER.
- MRC reads of the following registers:
  - PMCEID0 and PMCEID1.
  - If FEAT_PMUv3p1 is implemented, PMCEID2 and PMCEID3.
- MCR writes to PMSWINC.
- MRRC or MCRR accesses to PMCCNTR.

When trapped, reads and writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MRS and MSR accesses are reported using EC syndrome value 0x18.
- AArch32 MRC and MCR accesses are reported using EC syndrome value 0x03.
- AArch32 MRRC and MCRR accesses are reported using EC syndrome value 0x04.

This field is ignored by the PE when FEAT_PMUv3p9 is implemented and PMUSERENR_EL0.UEN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMUSERENR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMUSERENR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1110	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMUSERENR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMUSERENR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMUSERENR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMUSERENR_EL0;
```

```
MSR PMUSERENR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1110	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMUSERENR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMUSERENR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMUSERENR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMXEVCNTR_EL0, Performance Monitors Selected Event Count Register

The PMXEVCNTR_EL0 characteristics are:

Purpose

Reads or writes the value of the selected event counter, PMEVCNTR<n>_EL0. PMSELR_EL0.SEL determines which event counter is selected.

Configuration

AArch64 System register PMXEVCNTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMXEVCNTR[31:0].

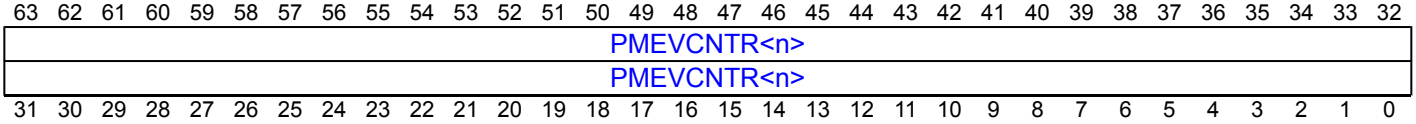
This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMXEVCNTR_EL0 are UNDEFINED.

Attributes

PMXEVCNTR_EL0 is a 64-bit register.

Field descriptions

When FEAT_PMUv3p5 is implemented:



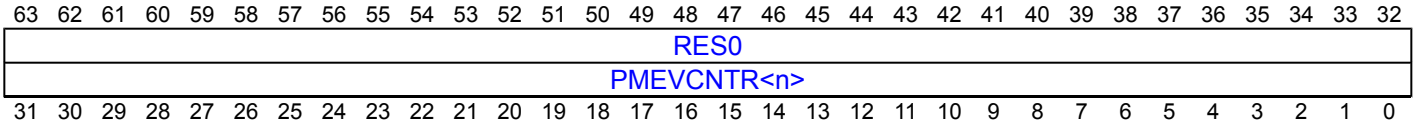
PMEVCNTR<n>, bits [63:0]

Value of the selected event counter, PMEVCNTR<n>_EL0, where n is the value stored in PMSELR_EL0.SEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:



Bits [63:32]

Reserved, RES0.

PMEVCNTR<n>, bits [31:0]

Value of the selected event counter, PMEVCNTR<n>_EL0, where n is the value stored in PMSELR_EL0.SEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMXEVCNTR_EL0

If FEAT_FGT is implemented and PMSELR_EL0.SEL is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMXEVCNTR_EL0 is as follows:

- If PMSELR_EL0.SEL is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and PMSELR_EL0.SEL is greater than or equal to the number of accessible event counters, then reads and writes of PMXEVCNTR_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if PMSELR_EL0.SEL has an UNKNOWN value less than the number of counters accessible at the current Exception level and Security state.
- If EL2 is implemented and enabled in the current Security state, and PMSELR_EL0.SEL is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMXEVCNTR_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.P<UInt(PMSELR_EL0.SEL)> == 0.

Permitted writes of PMXEVCNTR_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.{UEN,ER} == {1,1}.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.{UEN,ER,EN}.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible event counters.

Otherwise, the number of accessible event counters is determined by the Effective value of PMCCR.EPMN. For more information, see MDCR_EL2.HPMN and PMCCR.EPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMXEVCNTR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif UInt(PMSELR_EL0.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' &&
PMUACR_EL1[UInt(PMSELR_EL0.SEL)] == '0' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)];
```

```
MSR PMXEVCNTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif UInt(PMSELR_EL0.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' &&
(PMUACR_EL1[UInt(PMSELR_EL0.SEL)] == '0' || PMUSERENR_EL0.ER == '1') then
        return;
    else
        PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMXEVTYPER_EL0, Performance Monitors Selected Event Type Register

The PMXEVTYPER_EL0 characteristics are:

Purpose

When PMSELR_EL0.SEL selects an event counter, this accesses a PMEVTYPER<n>_EL0 register. When PMSELR_EL0.SEL selects the cycle counter, this accesses PMCCFILTR_EL0.

Configuration

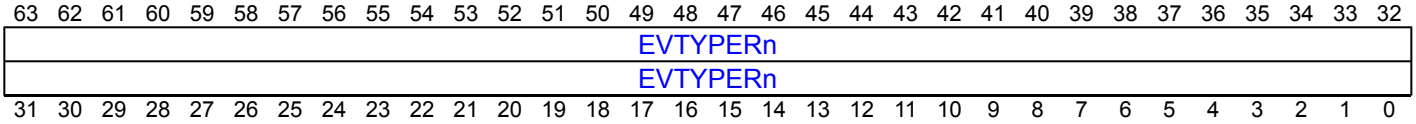
AArch64 System register PMXEVTYPER_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMXEVTYPER[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMXEVTYPER_EL0 are UNDEFINED.

Attributes

PMXEVTYPER_EL0 is a 64-bit register.

Field descriptions



EVTYPERn, bits [63:0]

When PMSELR_EL0.SEL == 31, this register accesses PMCCFILTR_EL0.

Otherwise, this register accesses PMEVTYPER<n>_EL0 where n is the value in PMSELR_EL0.SEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMXEVTYPER_EL0

If FEAT_FGT is implemented, and PMSELR_EL0.SEL is not 31 and is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMXEVTYPER_EL0 is as follows:

- If PMSELR_EL0.SEL is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented, and PMSELR_EL0.SEL is not 31 and is greater than or equal to the number of accessible event counters, then reads and writes of PMXEVTYPER_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if PMSELR_EL0.SEL has an UNKNOWN value less than the number of event counters accessible at the current Exception level and Security state.
- Accesses to the register behave as if PMSELR_EL0.SEL is 31.
- If EL2 is implemented and enabled in the current Security state, PMSELR_EL0 is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMXEVTYPER_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- Any of the following are true:
  - PMSELR_EL0.SEL != 31 and PMUACR_EL1.P<UInt(PMSELR_EL0.SEL)> == 0.
  - PMSELR_EL0.SEL == 31 and PMUACR_EL1.C == 0.

Permitted writes of PMXEVTYPER_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- Any of the following are true:
  - PMSELR_EL0.SEL != 31 and PMUSERENR_EL0.ER == 1.
  - PMSELR_EL0.SEL == 31 and PMUSERENR_EL0.CR == 1.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.{UEN,EN}.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible event counters.

Otherwise, the number of accessible event counters is determined by the Effective value of PMCCR.EPMN. For more information, see MDCR_EL2.HPMN and PMCCR.EPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMXEVTYPER_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >=
GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && ((UInt(PMSELR_EL0.SEL)
!= 31 && PMUACR_EL1[UInt(PMSELR_EL0.SEL)] == '0') || (UInt(PMSELR_EL0.SEL) == 31 && PMUACR_EL1.C
== '0')) then
        X[t, 64] = Zeros(64);
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        X[t, 64] = PMCCFILTR_EL0;
    else
        X[t, 64] = PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >=
GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        X[t, 64] = PMCCFILTR_EL0;
    else
        X[t, 64] = PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        X[t, 64] = PMCCFILTR_EL0;
    else
        X[t, 64] = PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL3 then
    if UInt(PMSELR_EL0.SEL) == 31 then
        X[t, 64] = PMCCFILTR_EL0;
    else
        X[t, 64] = PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)];
```

```
MSR PMXEVTYPER_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >=
GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && ((UInt(PMSELR_EL0.SEL)
!= 31 && (PMUACR_EL1[UInt(PMSELR_EL0.SEL)] == '0' || PMUSERENR_EL0.ER == '1')) ||
(UInt(PMSELR_EL0.SEL) == 31 && (PMUACR_EL1.C == '0' || PMUSERENR_EL0.CR == '1'))) then
        return;
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        PMCCFILTR_EL0 = X[t, 64];
    else
        PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >=
GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        PMCCFILTR_EL0 = X[t, 64];
    else
        PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        PMCCFILTR_EL0 = X[t, 64];
    else
        PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if UInt(PMSELR_EL0.SEL) == 31 then
        PMCCFILTR_EL0 = X[t, 64];
    else
        PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMZR_EL0, Performance Monitors Zero with Mask

The PMZR_EL0 characteristics are:

Purpose

Zero the set of counters specified by the mask written to PMZR_EL0.

Configuration

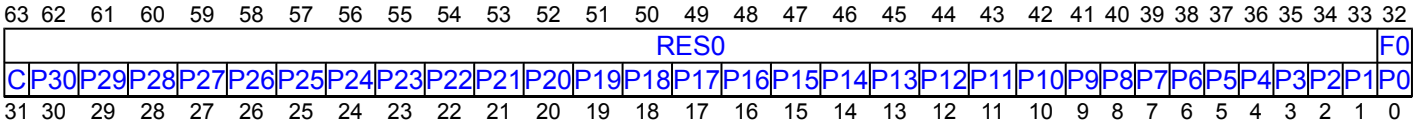
AArch64 System register PMZR_EL0 bits [63:0] are architecturally mapped to External register PMZR_EL0[63:0] when FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p9 is implemented.

This register is present only when FEAT_PMUv3p9 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMZR_EL0 are UNDEFINED.

Attributes

PMZR_EL0 is a 64-bit register.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Zero PMICNTR_EL0.

F0	Meaning
0b0	Write is ignored.
0b1	Set PMICNTR_EL0 to zero.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - PSTATE.EL != EL3.
  - MDCR_EL3.EnPM2 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 0 or PMUACR_EL1.F0 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_FGT2 is implemented.
  - EL2Enabled().
  - PSTATE.EL IN {EL1, EL0}.
  - HCR_EL2.[E2H,TGE] != 0b11.
  - (EL3 is implemented and SCR_EL3.FGTEn2 == 0) or HDFGWTR2_EL2.nPMICNTR_EL0 == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.IR == 1.

- Otherwise, access to this field is **WO/RAZ**.

Otherwise:

Reserved, RES0.

C, bit [31]

Zero PMCCNTR_EL0.

C	Meaning
0b0	Write is ignored.
0b1	Set PMCCNTR_EL0 to zero.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.C == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,CR] == 0b11.
- Otherwise, access to this field is **WO/RAZ**.

P<m>, bit [m], for m = 30 to 0

Zero PMEVCNTR<m>_EL0.

P<m>	Meaning
0b0	Write is ignored.
0b1	Set PMEVCNTR<m>_EL0 to zero.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.P<m> == 0.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - PMUSERENR_EL0.[UEN,ER] == 0b11.
- Otherwise, access to this field is **WO/RAZ**.

Accessing PMZR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MSR PMZR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b100

```
if !(IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN ==
'0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nPMZR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ZeroPMUCounters(X[t, 64]);
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nPMZR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ZeroPMUCounters(X[t, 64]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ZeroPMUCounters(X[t, 64]);
elsif PSTATE.EL == EL3 then
    ZeroPMUCounters(X[t, 64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

POR_EL0, Permission Overlay Register 0 (EL0)

The POR_EL0 characteristics are:

Purpose

Stage 1 Permission Overlay Register for unprivileged access of EL1&0 or EL2&0 translation regime.

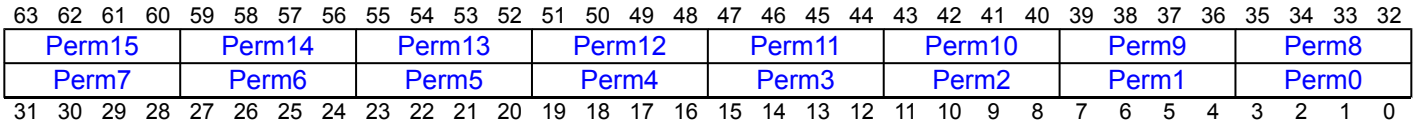
Configuration

This register is present only when FEAT_S1POE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to POR_EL0 are UNDEFINED.

Attributes

POR_EL0 is a 64-bit register.

Field descriptions



Perm<m>, bits [4m+3:4m], for m = 15 to 0

Perm Represents stage 1 Overlay Permissions.

Perm<m>	Meaning
0b0000	No access.
0b0001	Read.
0b0010	Execute.
0b0011	Read, Execute.
0b0100	Write.
0b0101	Write, Read.
0b0110	Write, Execute.
0b0111	Read, Write, Execute.
0b1xxx	Reserved - treated as No access

If FEAT_D128 is implemented and VMSAv9-128 is in use, then fields Perm[8] to Perm[15] are used for POIndex values 8 to 15.

Otherwise, the fields Perm[8] to Perm[15] are RES0.

This field is not permitted to be cached in a TLB.

When the stage 1 Overlay mechanism is disabled, this field is IGNORED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing POR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, POR_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && CPACR_EL1.E0POE == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGRTR_EL2.nPOR_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CPTR_EL2.E0POE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = POR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nPOR_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = POR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = POR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = POR_EL0;
```

MSR POR_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && CPACR_EL1.E0POE == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGWTR_EL2.nPOR_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CPTR_EL2.E0POE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        POR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nPOR_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        POR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        POR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    POR_EL0 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

POR_EL1, Permission Overlay Register 1 (EL1)

The POR_EL1 characteristics are:

Purpose

Stage 1 Permission Overlay Register for privileged access of the EL1&0 translation regime.

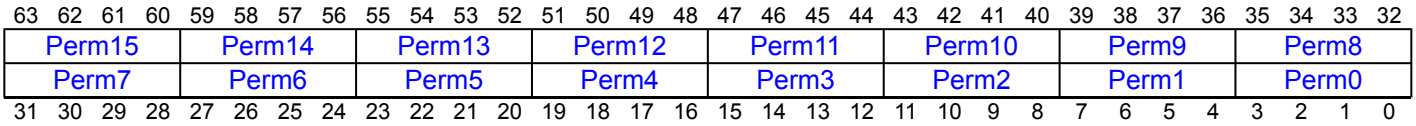
Configuration

This register is present only when FEAT_S1POE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to POR_EL1 are UNDEFINED.

Attributes

POR_EL1 is a 64-bit register.

Field descriptions



Perm<m>, bits [4m+3:4m], for m = 15 to 0

Perm Represents stage 1 Overlay Permissions.

Perm<m>	Meaning
0b0000	No access.
0b0001	Read.
0b0010	Execute.
0b0011	Read, Execute.
0b0100	Write.
0b0101	Write, Read.
0b0110	Write, Execute.
0b0111	Read, Write, Execute.
0b1xxx	Reserved - treated as No access

If FEAT_D128 is implemented and VMSAv9-128 is in use, then fields Perm[8] to Perm[15] are used for POIndex values 8 to 15.

Otherwise, the fields Perm[8] to Perm[15] are RES0.

This field is not permitted to be cached in a TLB.

When the stage 1 Overlay mechanism is disabled, this field is IGNORED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing POR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name POR_EL1 or POR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, POR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nPOR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x2A8];
    else
        X[t, 64] = POR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = POR_EL2;
    else
        X[t, 64] = POR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = POR_EL1;
```

MSR POR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nPOR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x2A8] = X[t, 64];
    else
        POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        POR_EL2 = X[t, 64];
    else
        POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    POR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, POR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x2A8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = POR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = POR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR POR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x2A8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            POR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        POR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

POR_EL2, Permission Overlay Register 2 (EL2)

The POR_EL2 characteristics are:

Purpose

Stage 1 Permission Overlay Register for privileged access of the EL2 or EL2&0 translation regime.

Configuration

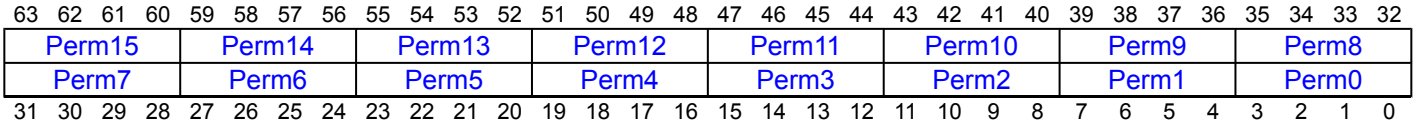
This register is present only when FEAT_S1POE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to POR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

POR_EL2 is a 64-bit register.

Field descriptions



Perm<m>, bits [4m+3:4m], for m = 15 to 0

Perm Represents stage 1 Overlay Permissions.

Perm<m>	Meaning
0b0000	No access.
0b0001	Read.
0b0010	Execute.
0b0011	Read, Execute.
0b0100	Write.
0b0101	Write, Read.
0b0110	Write, Execute.
0b0111	Read, Write, Execute.
0b1xxx	Reserved - treated as No access

If FEAT_D128 is implemented and VMSAv9-128 is in use, then fields Perm[8] to Perm[15] are used for POIndex values 8 to 15.

Otherwise, the fields Perm[8] to Perm[15] are RES0.

This field is not permitted to be cached in a TLB.

When the stage 1 Overlay mechanism is disabled, this field is IGNORED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing POR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name POR_EL2 or POR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, POR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = POR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = POR_EL2;
```

MSR POR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        POR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    POR_EL2 = X[t, 64];
```

MRS <Xt>, POR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nPOR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x2A8];
    else
        X[t, 64] = POR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = POR_EL2;
    else
        X[t, 64] = POR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = POR_EL1;
```

```
MSR POR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nPOR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x2A8] = X[t, 64];
    else
        POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        POR_EL2 = X[t, 64];
    else
        POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    POR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

POR_EL3, Permission Overlay Register 3 (EL3)

The POR_EL3 characteristics are:

Purpose

Stage 1 Permission Overlay Register for privileged access of the EL3 translation regime.

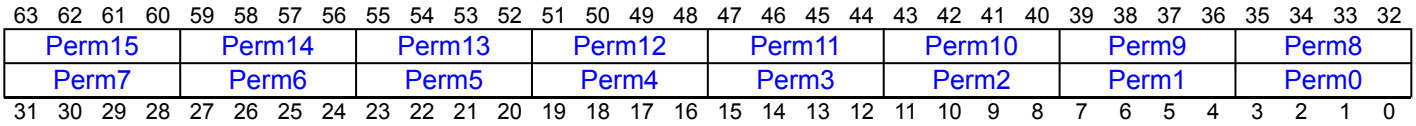
Configuration

This register is present only when FEAT_S1POE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to POR_EL3 are UNDEFINED.

Attributes

POR_EL3 is a 64-bit register.

Field descriptions



Perm<m>, bits [4m+3:4m], for m = 15 to 0

Perm Represents stage 1 Overlay Permissions.

Perm<m>	Meaning
0b0000	No access.
0b0001	Read.
0b0010	Execute.
0b0011	Read, Execute.
0b0100	Write.
0b0101	Write, Read.
0b0110	Write, Execute.
0b0111	Read, Write, Execute.
0b1xxx	Reserved - treated as No access

If FEAT_D128 is implemented and VMSAv9-128 is in use, then fields Perm[8] to Perm[15] are used for POIndex values 8 to 15.

Otherwise, the fields Perm[8] to Perm[15] are RES0.

This field is not permitted to be cached in a TLB.

When stage 1 Overlay mechanism is disabled, this register is ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing POR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, POR_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = POR_EL3;
```

MSR POR_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    POR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RCWMASK_EL1, Read Check Write Instruction Mask (EL1)

The RCWMASK_EL1 characteristics are:

Purpose

Contains the mask used by RCW instructions.

Configuration

This register is present only when FEAT_THE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to RCWMASK_EL1 are UNDEFINED.

RCWMASK_EL1 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

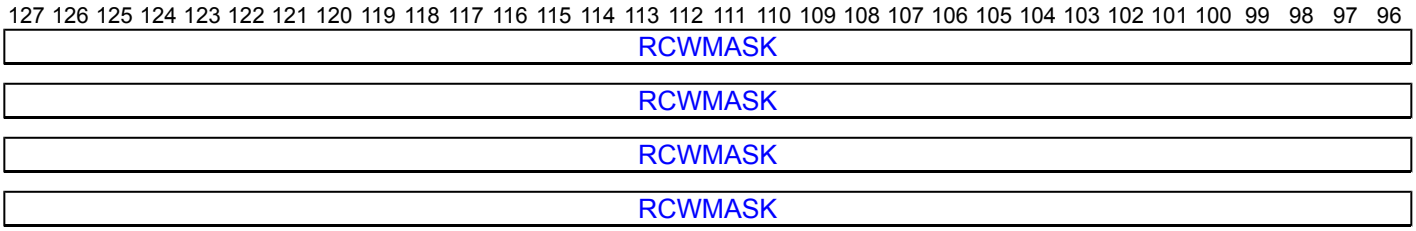
Attributes

RCWMASK_EL1 is a:

- 128-bit register when FEAT_D128 is implemented
- 64-bit register otherwise

Field descriptions

When FEAT_D128 is implemented:



RCWMASK, bits [127:0]

Mask used to decide which bit-fields are writable to the 128-bit Descriptor by RCW or RCWS instructions.

If RCWMASK_EL1 is indirectly read by 128-bit variants of RCW or RCWS instructions:

- The Effective value of RCWMASK[n] is the same as RCWMASK_EL1[n], except as follows:
  - If n >= 17, and n <= 55, the Effective value of RCWMASK[n] is the same as RCWMASK_EL1[16].
  - If n is in {126:125, 120:119, 114, 107:101, 90:56, 1:0}, the Effective value of RCWMASK[n] is 0.
  - If n >= 121, n <= 124, and FEAT_S1POE is not implemented, the Effective value of RCWMASK[n] is 0.
  - If FEAT_MEC is not implemented, the Effective value of RCWMASK[108] is 0.

If RCWMASK_EL1 is indirectly read by 64-bit variants of RCW or RCWS instructions:

- The Effective value of RCWMASK[n] is the same as RCWMASK_EL1[n], except as follows:
  - If n >= 18, and n <= 49, the Effective value of RCWMASK[n] is the same as RCWMASK_EL1[17].

RCWMASK_EL1 register bits {126:125, 120:119, 114, 107:101, 90:64, 52, 49:18, 0} are RES0.

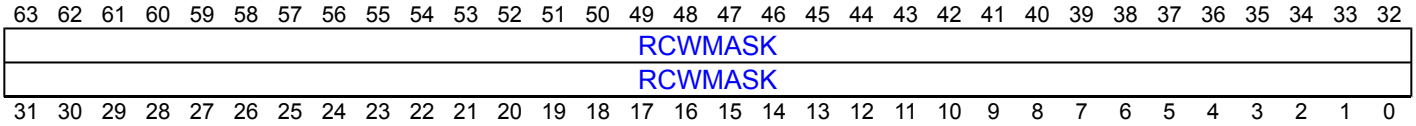
If FEAT_S1POE is not implemented, RCWMASK_EL1 register bits {124:121} are RES0.

If FEAT_MEC is not implemented, RCWMASK_EL1[108] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:



RCWMASK, bits [63:0]

Mask used to decide which bit-fields are writable to the 64-bit Descriptor by RCW or RCWS Instructions.

The Effective value of RCWMASK[n] is the same as RCWMASK_EL1[n], except as follows:

- If n >= 18, and n <= 49, the Effective value of RCWMASK[n] is the same as RCWMASK_EL1[17].

RCWMASK_EL1 register bits {52, 49:18, 0} are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RCWMASK_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RCWMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nRCWMASK_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RCWMASK_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RCWMASK_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = RCWMASK_EL1<63:0>;
```

MSR RCWMASK_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nRCWMASK_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RCWMASK_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RCWMASK_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    RCWMASK_EL1<63:0> = X[t, 64];
```

When FEAT_D128 is implemented

```
MRRS <Xt>, <Xt+1>, RCWMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nRCWMASK_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = RCWMASK_EL1<127:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = RCWMASK_EL1<127:0>;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = RCWMASK_EL1<127:0>;
```

When FEAT_D128 is implemented

```
MSRR RCWMASK_EL1, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nRCWMASK_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        RCWMASK_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        RCWMASK_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    RCWMASK_EL1<127:0> = X[t, t2, 128];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RCWSMASK_EL1, Software Read Check Write Instruction Mask (EL1)

The RCWSMASK_EL1 characteristics are:

Purpose

Contains the software mask used by RCWS instructions.

Configuration

This register is present only when FEAT_THE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to RCWSMASK_EL1 are UNDEFINED.

RCWSMASK_EL1 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

Attributes

RCWSMASK_EL1 is a:

- 128-bit register when FEAT_D128 is implemented
- 64-bit register otherwise

Field descriptions

When FEAT_D128 is implemented:

127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
RCWSMASK																															
RCWSMASK																															
RCWSMASK																															
RCWSMASK																															

RCWSMASK, bits [127:0]

Software Mask used to decide which bit-fields are writable to the 128-bit Descriptor by RCWS instructions.

If RCWSMASK_EL1 is indirectly read by 128-bit variants of RCWS instructions:

- The Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[n], except as follows:
  - If n >= 17, and n <= 55, the Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[16].
  - If n is in {126:125, 120:119, 114, 107:101, 90:56, 1:0}, the Effective value of RCWSMASK[n] is 0.
  - If n >= 121, n <= 124, and FEAT_S1POE is not implemented, the Effective value of RCWSMASK[n] is 0.
  - If FEAT_MEC is not implemented, the Effective value of RCWSMASK bit RCWSMASK_EL1[108] is 0.

If RCWSMASK_EL1 is indirectly read by 64-bit variants of RCWS instructions:

- The Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[n], except as follows:
  - If n >= 18, and n <= 49, the Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[17].

- If n == 52 and Protection is enabled, the Effective value of RCWSMASK[52] is 0.

RCWSMASK_EL1 register bits {126:125, 120:119, 114, 107:101, 90:64, 49:18, 0} are RES0.

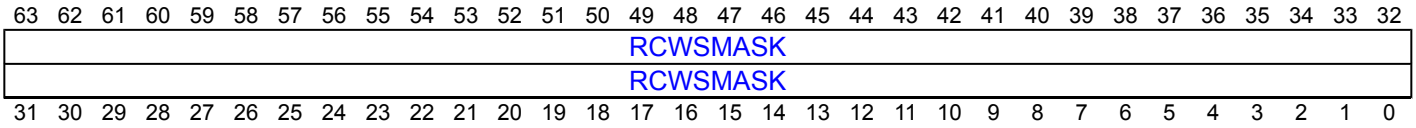
If FEAT_S1POE is not implemented, RCWSMASK_EL1 register bits {124:121} are RES0.

If FEAT_MEC is not implemented, RCWSMASK_EL1[108] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:



RCWSMASK, bits [63:0]

Software Mask used to decide which bit-fields are writable to the 64-bit Descriptor by RCWS Instruction.

The Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[n], except as follows

- If n >= 18, and n <= 49, the Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[17].
- If n == 52 and Protection is enabled, the Effective value of RCWSMASK[52] is 0.

RCWSMASK_EL1 register bits {49:18, 0} are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RCWSMASK_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RCWSMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nRCWSMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RCWSMASK_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RCWSMASK_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = RCWSMASK_EL1<63:0>;
```

```
MSR RCWSMASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nRCWSMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RCWSMASK_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RCWSMASK_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    RCWSMASK_EL1<63:0> = X[t, 64];
```

When FEAT_D128 is implemented

```
MRRS <Xt>, <Xt+1>, RCWSMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nRCWSMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = RCWSMASK_EL1<127:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = RCWSMASK_EL1<127:0>;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = RCWSMASK_EL1<127:0>;
```

When FEAT_D128 is implemented

```
MSRR RCWSMASK_EL1, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nRCWSMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        RCWSMASK_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        RCWSMASK_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    RCWSMASK_EL1<127:0> = X[t, t2, 128];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

REVIDR_EL1, Revision ID Register

The REVIDR_EL1 characteristics are:

Purpose

Provides implementation-specific minor revision information.

Configuration

AArch64 System register REVIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register REVIDR[31:0].

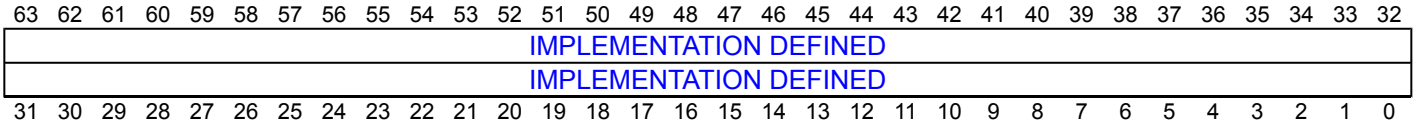
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to REVIDR_EL1 are UNDEFINED.

If REVIDR_EL1 has the same value as MIDR_EL1, then its contents have no significance.

Attributes

REVIDR_EL1 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing REVIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, REVIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0000	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.REVIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = REVIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = REVIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = REVIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RGSR_EL1, Random Allocation Tag Seed Register.

The RGSR_EL1 characteristics are:

Purpose

Random Allocation Tag Seed Register.

Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to RGSR_EL1 are UNDEFINED.

When GCR_EL1.RRND==0b1, updates to RGSR_EL1 are implementation-specific.

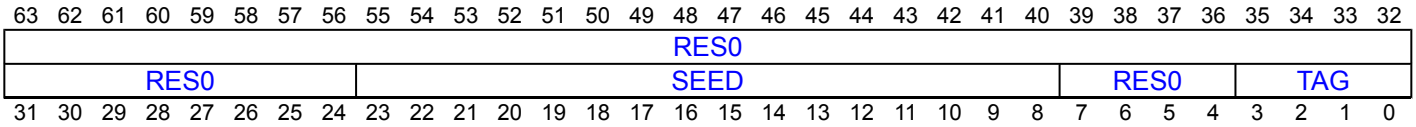
Direct and indirect reads and writes to the register appear to occur in program order relative to other instructions, without the need for any explicit synchronization.

Attributes

RGSR_EL1 is a 64-bit register.

Field descriptions

When GCR_EL1.RRND == 0:



Bits [63:24]

Reserved, RES0.

SEED, bits [23:8]

Seed register used for generating values returned by RandomTag().

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

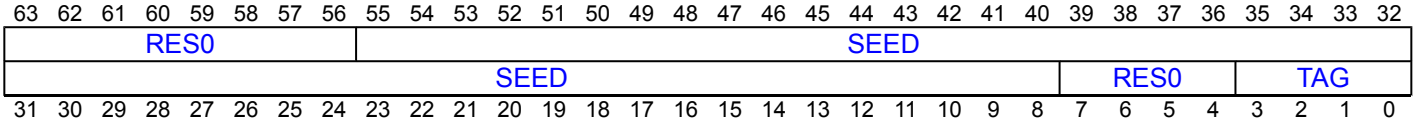
TAG, bits [3:0]

Tag generated by the most recent IRG instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:



Bits [63:56]

Reserved, RES0.

SEED, bits [55:8]

IMPLEMENTATION DEFINED.

Note

Software is recommended to avoid writing SEED[15:0] with a value of zero, unless this has been generated by the PE in response to an earlier value with SEED being nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

TAG, bits [3:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RGSR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RGSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RGSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RGSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = RGSR_EL1;
```

```
MSR RGSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RGSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RGSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    RGSR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RMR_EL1, Reset Management Register (EL1)

The RMR_EL1 characteristics are:

Purpose

When this register is implemented:

- A write to the register at EL1 can request a Warm reset.
- If EL1 can use all Execution states, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch64 System register RMR_EL1 bits [31:0] are architecturally mapped to AArch32 System register RMR[31:0] when the highest implemented Exception level is EL1.

This register is present only when the highest implemented Exception level is EL1 and FEAT_AA64 is implemented. Otherwise, direct accesses to RMR_EL1 are UNDEFINED.

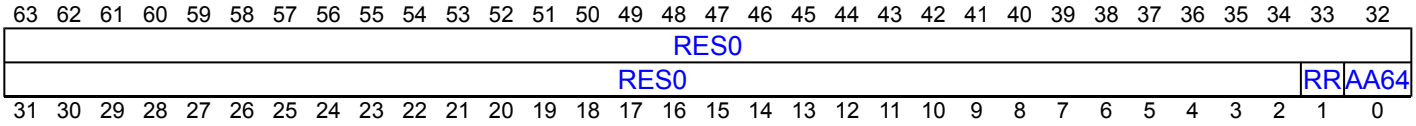
When EL1 is the highest implemented Exception level:

- If EL1 can use all Execution states then this register must be implemented.
- If EL1 cannot use AArch32 then it is IMPLEMENTATION DEFINED whether the register is implemented.

Attributes

RMR_EL1 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

AA64, bit [0]

When EL1 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

AA64	Meaning
0b0	AArch32.
0b1	AArch64.

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL1 can only use AArch64 state, this bit is RAO/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'1'`.

Accessing RMR_EL1

Accesses to this register use the following encodings in the System register encoding space:

`MRS <Xt>, RMR_EL1`

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b010

```
if !(IsHighestEL(EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL1) then
    X[t, 64] = RMR_EL1;
else
    UNDEFINED;
```

`MSR RMR_EL1, <Xt>`

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b010

```
if !(IsHighestEL(EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL1) then
    RMR_EL1 = X[t, 64];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RMR_EL2, Reset Management Register (EL2)

The RMR_EL2 characteristics are:

Purpose

When this register is implemented:

- A write to the register at EL2 can request a Warm reset.
- If EL2 can use all Execution states, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch64 System register RMR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HRMR[31:0] when the highest implemented Exception level is EL2.

This register is present only when the highest implemented Exception level is EL2 and FEAT_AA64 is implemented. Otherwise, direct accesses to RMR_EL2 are UNDEFINED.

When EL2 is the highest implemented Exception level:

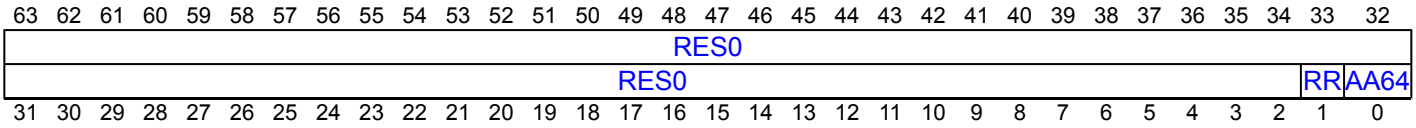
- If EL2 can use all Execution states then this register must be implemented.
- If EL2 cannot use AArch32 then it is IMPLEMENTATION DEFINED whether the register is implemented.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

RMR_EL2 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

AA64, bit [0]

When EL2 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

AA64	Meaning
`0b0`	AArch32.
`0b1`	AArch64.

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL2 can only use AArch64 state, this bit is RAO/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'1'`.

Accessing RMR_EL2

Accesses to this register use the following encodings in the System register encoding space:

`MRS <Xt>, RMR_EL2`

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0000	0b010

```
if !(IsHighestEL(EL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL2) && EffectiveHCR_EL2_NVx() IN {'xx1'} then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    X[t, 64] = RMR_EL2;
else
    UNDEFINED;
```

`MSR RMR_EL2, <Xt>`

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0000	0b010

```
if !(IsHighestEL(EL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL2) && EffectiveHCR_EL2_NVx() IN {'xx1'} then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    RMR_EL2 = X[t, 64];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RMR_EL3, Reset Management Register (EL3)

The RMR_EL3 characteristics are:

Purpose

If EL3 is implemented and this register is implemented:

- A write to the register at EL3 can request a Warm reset.
- If EL3 can use all Execution states, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch64 System register RMR_EL3 bits [31:0] are architecturally mapped to AArch32 System register RMR[31:0] when EL3 is implemented.

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to RMR_EL3 are UNDEFINED.

When EL3 is implemented:

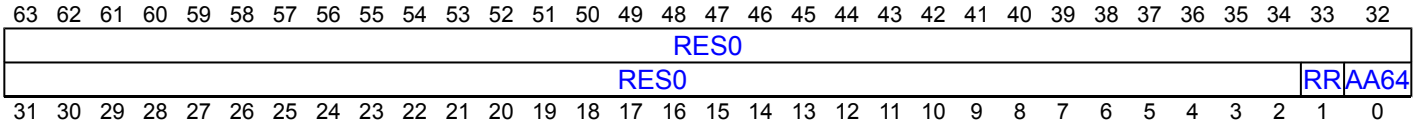
- If EL3 can use all Execution states then this register must be implemented.
- If EL3 cannot use AArch32, then it is IMPLEMENTATION DEFINED whether the register is implemented.

Otherwise, direct accesses to RMR_EL3 are UNDEFINED.

Attributes

RMR_EL3 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

AA64, bit [0]

When EL3 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

AA64	Meaning
0b0	AArch32.
0b1	AArch64.

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL3 can only use AArch64 state, this bit is RAO/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'1'`.

Accessing RMR_EL3

Accesses to this register use the following encodings in the System register encoding space:

`MRS <Xt>, RMR_EL3`

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL3 && IsHighestEL(EL3) then
    X[t, 64] = RMR_EL3;
else
    UNDEFINED;
```

`MSR RMR_EL3, <Xt>`

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL3 && IsHighestEL(EL3) then
    RMR_EL3 = X[t, 64];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RNDR, Random Number

The RNDR characteristics are:

Purpose

Random Number. Returns a 64-bit random number from an approved Random Bit Generator, where the Deterministic Random Bit Generator within the Random Bit Generator is reseeded from an approved entropy source at an IMPLEMENTATION DEFINED rate. See 'Properties of the generated random number'.

If the hardware returns a genuine random number, PSTATE.NZCV is set to `0b0000`.

If the instruction cannot return a genuine random number in a reasonable period of time, PSTATE.NZCV is set to `0b0100` and the data value returned is 0.

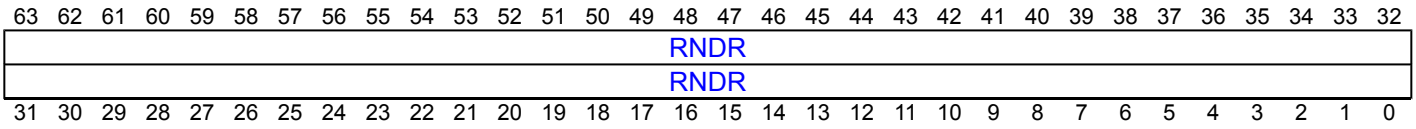
Configuration

This register is present only when (FEAT_RNG is implemented or FEAT_RNG_TRAP is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to RNDR are UNDEFINED.

Attributes

RNDR is a 64-bit register.

Field descriptions



RNDR, bits [63:0]

Random Number. Returns a 64-bit Random Number from an approved Random Bit Generator, where the Deterministic Random Bit Generator within the Random Bit Generator is reseeded from an approved entropy source at an IMPLEMENTATION DEFINED rate. See 'Properties of the generated random number'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RNDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RNDR
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0010	0b0100	0b000

```
if !((IsFeatureImplemented(FEAT_RNG) || IsFeatureImplemented(FEAT_RNG_TRAP)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDR;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDR;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDR;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RNDRRS, Random Number Full Entropy

The RNDRRS characteristics are:

Purpose

Random Number with fresh full entropy. Returns a 64-bit random number from an approved Random Bit Generator, using either a Non-deterministic Random Bit Generator or one where the Deterministic Random Bit Generator is reseeded, where possible, from an approved entropy source before the return of the random number. See 'Properties of the generated random number'.

If the hardware returns a genuine random number, PSTATE.NZCV is set to `0b0000`.

If the instruction cannot return a genuine random number in a reasonable period of time, PSTATE.NZCV is set to `0b0100` and the data value returned is 0.

When FEAT_RNG_TRAP is implemented and SCR_EL3.TRNDR is 1, reads of this register are trapped to EL3.

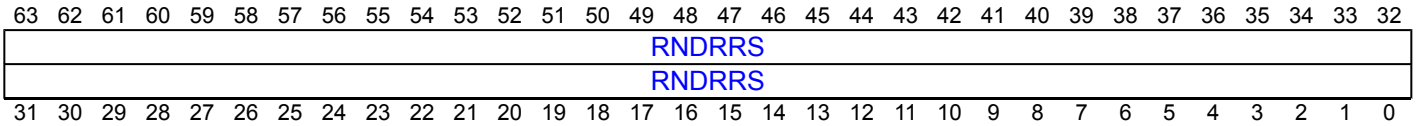
Configuration

This register is present only when (FEAT_RNG is implemented or FEAT_RNG_TRAP is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to RNDRRS are UNDEFINED.

Attributes

RNDRRS is a 64-bit register.

Field descriptions



RNDRRS, bits [63:0]

Random Number with fresh full entropy. Returns a 64-bit random number from an approved Random Bit Generator, using either a Non-deterministic Random Bit Generator or one where the Deterministic Random Bit Generator is reseeded, where possible, from an approved entropy source before the return of the random number. See 'Properties of the generated random number'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RNDRRS

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RNDRRS
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0010	0b0100	0b001

```
if !((IsFeatureImplemented(FEAT_RNG) || IsFeatureImplemented(FEAT_RNG_TRAP)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDRRS;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDRRS;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDRRS;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDRRS;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RVBAR_EL1, Reset Vector Base Address Register (if EL2 and EL3 not implemented)

The RVBAR_EL1 characteristics are:

Purpose

If EL1 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

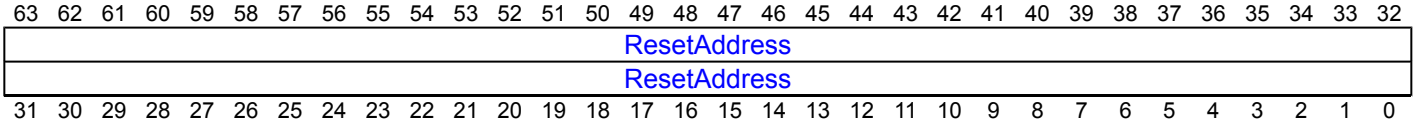
Configuration

This register is present only when the highest implemented Exception level is EL1 and FEAT_AA64 is implemented. Otherwise, direct accesses to RVBAR_EL1 are UNDEFINED.

Attributes

RVBAR_EL1 is a 64-bit register.

Field descriptions



ResetAddress, bits [63:0]

The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing RVBAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RVBAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b001

```
if !(IsHighestEL(EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL1) then
    X[t, 64] = RVBAR_EL1;
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RVBAR_EL2, Reset Vector Base Address Register (if EL3 not implemented)

The RVBAR_EL2 characteristics are:

Purpose

If EL2 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

Configuration

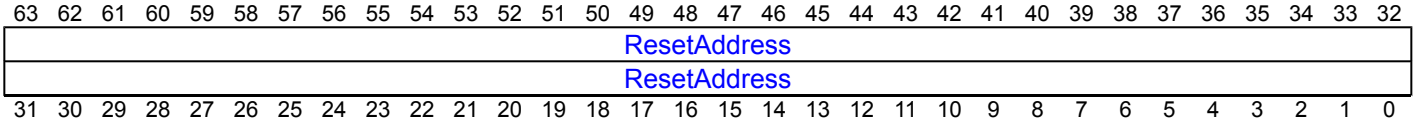
This register is present only when the highest implemented Exception level is EL2 and FEAT_AA64 is implemented. Otherwise, direct accesses to RVBAR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

RVBAR_EL2 is a 64-bit register.

Field descriptions



ResetAddress, bits [63:0]

The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing RVBAR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RVBAR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0000	0b001

```
if !(IsHighestEL(EL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL2) && EffectiveHCR_EL2_NVx() IN {'xx1'} then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    X[t, 64] = RVBAR_EL2;
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RVBAR_EL3, Reset Vector Base Address Register (if EL3 implemented)

The RVBAR_EL3 characteristics are:

Purpose

If EL3 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

Configuration

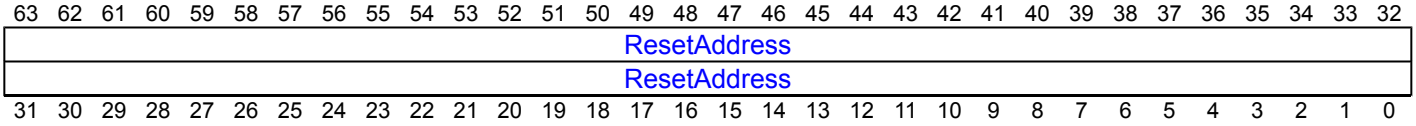
This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to RVBAR_EL3 are UNDEFINED.

Only implemented if the highest Exception level implemented is EL3.

Attributes

RVBAR_EL3 is a 64-bit register.

Field descriptions



ResetAddress, bits [63:0]

The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing RVBAR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RVBAR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0000	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL3 && IsHighestEL(EL3) then
    X[t, 64] = RVBAR_EL3;
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2>, SYSP S1_<op1>_<Cn>_<Cm>_<op2>, IMPLEMENTATION DEFINED System instructions

The SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2>, SYSP S1_<op1>_<Cn>_<Cm>_<op2> characteristics are:

Purpose

This area of the System instruction encoding space is reserved for IMPLEMENTATION DEFINED System instructions.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2>, SYSP S1_<op1>_<Cn>_<Cm>_<op2> are UNDEFINED.

Attributes

SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2>, SYSP S1_<op1>_<Cn>_<Cm>_<op2> is a:

- 128-bit System instruction when FEAT_SYSINSTR128 is implemented
- 64-bit System instruction otherwise

Field descriptions

When FEAT_SYSINSTR128 is implemented:

127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
IMPLEMENTATION DEFINED																															
IMPLEMENTATION DEFINED																															
IMPLEMENTATION DEFINED																															
IMPLEMENTATION DEFINED																															

IMPLEMENTATION DEFINED, bits [127:0]

IMPLEMENTATION DEFINED.

Otherwise:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
IMPLEMENTATION DEFINED																															
IMPLEMENTATION DEFINED																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Executing SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2>, SYSP S1_<op1>_<Cn>_<Cm>_<op2>

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
SYS #<op1>, <Cn>, <Cm>, #<op2>{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysInstr(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysInstr(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL2 then
    AArch64.ImpDefSysInstr(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysInstr(op0, op1, CRn, CRm, op2, t);
```

```
SYSL <Xt>, #<op1>, <Cn>, <Cm>, #<op2>
```

op0	op1	CRn	CRm	op2
0b01	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysInstrWithResult(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysInstrWithResult(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL2 then
    AArch64.ImpDefSysInstrWithResult(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysInstrWithResult(op0, op1, CRn, CRm, op2, t);
```

When FEAT_SYSINSTR128 is implemented

```
SYSP #<op1>, <Cn>, <Cm>, #<op2>{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        else
            AArch64.SystemAccessTrap(EL1, 0x14);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.ImpDefSysInstr128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.ImpDefSysInstr128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL2 then
    AArch64.ImpDefSysInstr128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysInstr128(op0, op1, CRn, CRm, op2, t, t2);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

S2PIR_EL2, Stage 2 Permission Indirection Register (EL2)

The S2PIR_EL2 characteristics are:

Purpose

Stage 2 Permission Indirection Register for EL1&0 translation regime.

Configuration

This register is present only when FEAT_S2PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to S2PIR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

S2PIR_EL2 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Perm15				Perm14				Perm13				Perm12				Perm11				Perm10				Perm9				Perm8			
Perm7				Perm6				Perm5				Perm4				Perm3				Perm2				Perm1				Perm0			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Perm<m>, bits [4m+3:4m], for m = 15 to 0

Represents stage 2 Base Permissions.

Perm<m>	Meaning
0b0000	No Access.
0b0001	Reserved - treated as No Access.
0b0010	MRO.
0b0011	MRO-TL1.
0b0100	WO.
0b0101	Reserved - treated as No Access.
0b0110	MRO-TL0.
0b0111	MRO-TL01.
0b1000	RO.
0b1001	RO+uX.
0b1010	RO+pX.
0b1011	RO+puX.
0b1100	RW.
0b1101	RW+uX.
0b1110	RW+pX.
0b1111	RW+puX.

This field is permitted to be cached in a TLB.

When stage 2 Indirect Permission mechanism is disabled, the contents of this register are ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing S2PIR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, S2PIR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_S2PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x2B0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = S2PIR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = S2PIR_EL2;
```

```
MSR S2PIR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_S2PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x2B0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        S2PIR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    S2PIR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

S2POR_EL1, Stage 2 Permission Overlay Register (EL1)

The S2POR_EL1 characteristics are:

Purpose

Stage 2 Permission Overlay Register for EL1&0 translation regime.

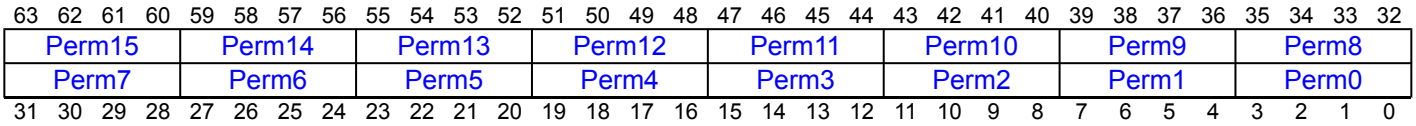
Configuration

This register is present only when FEAT_S2POE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to S2POR_EL1 are UNDEFINED.

Attributes

S2POR_EL1 is a 64-bit register.

Field descriptions



Perm<m>, bits [4m+3:4m], for m = 15 to 0

Configures stage 2 Overlay Permissions.

Perm<m>	Meaning
0b0000	No Access.
0b0001	Reserved - treated as No Access.
0b0010	MRO.
0b0011	MRO-TL1.
0b0100	WO.
0b0101	Reserved - treated as No Access.
0b0110	MRO-TL0.
0b0111	MRO-TL01.
0b1000	RO.
0b1001	RO+uX.
0b1010	RO+pX.
0b1011	RO+puX.
0b1100	RW.
0b1101	RW+uX.
0b1110	RW+pX.
0b1111	RW+puX.

This field is not permitted to be cached in a TLB.

When stage 2 Permission Overlay mechanism is disabled, this register is ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing S2POR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, S2POR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_S2POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nS2POR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x2B8];
    else
        X[t, 64] = S2POR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = S2POR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = S2POR_EL1;
```

MSR S2POR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_S2POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nS2POR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x2B8] = X[t, 64];
    else
        S2POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        S2POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    S2POR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

S3_<op1>_<Cn>_<Cm>_<op2>, IMPLEMENTATION DEFINED Registers

The S3_<op1>_<Cn>_<Cm>_<op2> characteristics are:

Purpose

This area of the instruction set space is reserved for IMPLEMENTATION DEFINED registers.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to S3_<op1>_<Cn>_<Cm>_<op2> are UNDEFINED.

When FEAT_SYSREG128 is implemented, each register in this space is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

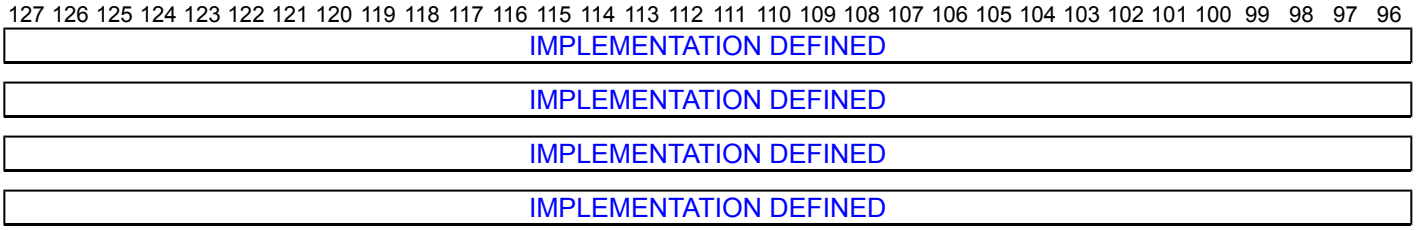
Attributes

S3_<op1>_<Cn>_<Cm>_<op2> is a:

- 128-bit register when FEAT_SYSREG128 is implemented
- 64-bit register otherwise

Field descriptions

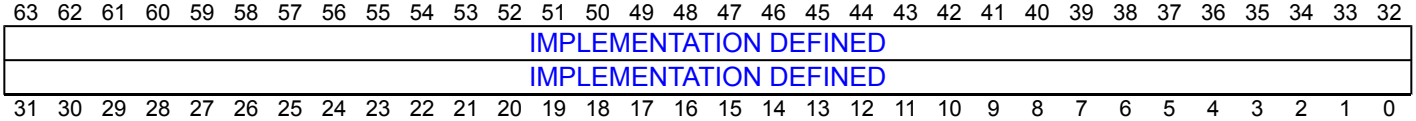
When FEAT_SYSREG128 is implemented:



IMPLEMENTATION DEFINED, bits [127:0]

IMPLEMENTATION DEFINED.

Otherwise:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing S3_<op1>_<Cn>_<Cm>_<op2>

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, S3_<op1>_C<Cn>_C<Cm>_<op2>
```

op0	op1	CRn	CRm	op2
0b11	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysRegRead(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysRegRead(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL2 then
    AArch64.ImpDefSysRegRead(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysRegRead(op0, op1, CRn, CRm, op2, t);
```

```
MSR S3_<op1>_C<Cn>_C<Cm>_<op2>, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysRegWrite(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysRegWrite(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL2 then
    AArch64.ImpDefSysRegWrite(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysRegWrite(op0, op1, CRn, CRm, op2, t);
```

When FEAT_SYSREG128 is implemented

```
MRRS <Xt>, <Xt+1>, S3_<op1>_C<Cn>_C<Cm>_<op2>
```

op0	op1	CRn	CRm	op2
0b11	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        else
            AArch64.SystemAccessTrap(EL1, 0x14);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif !ELIsInHost(EL0) && (!IsSCTLR2EL1Enabled() || SCTLR2_EL1.EnIDCP128 == '0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        else
            AArch64.SystemAccessTrap(EL1, 0x14);
    elsif ELIsInHost(EL0) && (!IsSCTLR2EL2Enabled() || SCTLR2_EL2.EnIDCP128 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && !ELIsInHost(EL0) && (!IsHCRXEL2Enabled() || HCRX_EL2.EnIDCP128 == '0')
then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        AArch64.ImpDefSysRegRead128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.EnIDCP128 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        AArch64.ImpDefSysRegRead128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        AArch64.ImpDefSysRegRead128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysRegRead128(op0, op1, CRn, CRm, op2, t, t2);
```

When FEAT_SYSREG128 is implemented

```
MSRR S3_<op1>_C<Cn>_C<Cm>_<op2>, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        else
            AArch64.SystemAccessTrap(EL1, 0x14);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif !ELIsInHost(EL0) && (!IsSCTLR2EL1Enabled() || SCTLR2_EL1.EnIDCP128 == '0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        else
            AArch64.SystemAccessTrap(EL1, 0x14);
    elsif ELIsInHost(EL0) && (!IsSCTLR2EL2Enabled() || SCTLR2_EL2.EnIDCP128 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && !ELIsInHost(EL0) && (!IsHCRXEL2Enabled() || HCRX_EL2.EnIDCP128 == '0')
then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        AArch64.ImpDefSysRegWrite128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.EnIDCP128 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        AArch64.ImpDefSysRegWrite128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        AArch64.ImpDefSysRegWrite128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysRegWrite128(op0, op1, CRn, CRm, op2, t, t2);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCR_EL3, Secure Configuration Register

The SCR_EL3 characteristics are:

Purpose

Defines the configuration of the current Security state. It specifies:

- The Security state of EL0, EL1, and EL2. The Security state is Secure, Non-secure, or Realm.
- The Execution state at lower Exception levels.
- Whether IRQ, FIQ, SError exceptions, and External abort exceptions are taken to EL3.
- Whether various operations are trapped to EL3.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCR_EL3 are UNDEFINED.

Attributes

SCR_EL3 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
RES0	NSE	HACDBSEn	HDBSSEn	FGTEn2	EnDSE	DSE	RES0	EnIDCP128	SRMASKEn	PFAREn	TWERR	TMEA	EnFPM	MECEn
TWEDEL		TWEDEn	ECVEn	FGTEn	ATA	EnSCXT	RES0	TID5	TID3	FIEN	NMEA	EASE	EEL2	API
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17

Bit [63]

Reserved, RES0.

NSE, bit [62]

When FEAT_RME is implemented:

This field, evaluated with SCR_EL3.NS, selects the Security state of EL2 and lower Exception levels.

For a description of the values derived by evaluating NS and NSE together, see SCR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

The Effective value of this bit is `0b0`.

Access to this field is **RES0**.

HACDBSEn, bit [61]

When FEAT_HACDBS is implemented:

Enables access to the HACDBSBR_EL2 and HACDBSCONS_EL2 registers at EL2.

HACDBSEn	Meaning
0b0	EL2 accesses to the specified registers are trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

Traps are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HDBSSEn, bit [60]
When FEAT_HDBSS is implemented:

Enables access to HDBSSBR_EL2 and HDBSSPROD_EL2 registers at EL2.

HDBSSEn	Meaning
0b0	EL2 accesses to the specified registers are trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

Traps are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FGTEn2, bit [59]
When FEAT_FGT2 is implemented:

Fine-Grained Traps Enable 2.

When EL2 is implemented, enables the traps to EL2 controlled by HDFGRTR2_EL2, HDFGWTR2_EL2, HFGITR2_EL2, HFGRTR2_EL2, and HFGWTR2_EL2, and controls access to those registers.

FGTEn2	Meaning
0b0	EL2 accesses to the specified registers are trapped to EL3. The values in these registers are treated as 0.
0b1	EL2 accesses to the specified registers are not trapped to EL3 by this mechanism.

Traps caused by accesses to the fine-grained trap registers are reported using EC syndrome value 0x18 and its associated ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnDSE, bit [58]

When FEAT_E3DSE is implemented:

Enable for delegated SError exceptions pended by SCR_EL3.DSE.

EnDSE	Meaning
0b0	Delegated SError exceptions pended by SCR_EL3.DSE are disabled.
0b1	Delegated SError exceptions pended by SCR_EL3.DSE are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSE, bit [57]

When FEAT_E3DSE is implemented:

Delegated SError exception for EL2, EL1, and EL0.

DSE	Meaning
0b0	This mechanism is not making a delegated SError exception pending.
0b1	A delegated SError exception for EL2, EL1, and EL0 is pending because of this mechanism.

When EL2 is implemented and enabled in the current Security state, delegated SError exceptions pended by this field are affected by HCR_EL2.AMO and HCRX_EL2.TMEA.

Virtual SError exceptions pended by HCR_EL2.VSE have priority over delegated SError exceptions pended by this field.

This field is ignored by the PE and treated as zero when SCR_EL3.EnDSE == 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [56]

Reserved, RES0.

EnIDCP128, bit [55]

When FEAT_SYSREG128 is implemented:

Enables access to IMPLEMENTATION DEFINED 128-bit System registers.

EnIDCP128	Meaning
0b0	Accesses at EL2, EL1, EL0 to IMPLEMENTATION DEFINED 128-bit System registers are trapped to EL3 using EC syndrome value 0x14, unless the access generates a higher priority exception. Disables the functionality of the 128-bit IMPLEMENTATION DEFINED System registers that are accessible at EL3.
0b1	No accesses are trapped by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SRMASKEn, bit [54]

Enables access to the following MASK registers:

- CPACRMASK_EL1, and CPACRMASK_EL12.
- SCTLRMASK_EL1, and SCTLRMASK_EL12.
- SCTLR2MASK_EL1, and SCTLR2MASK_EL12.
- TCRMASK_EL1, and TCRMASK_EL12.
- TCR2MASK_EL1, and TCR2MASK_EL12.
- ACTLRMASK_EL1 and ACTLRMASK_EL12, if they are implemented.
- CPTRMASK_EL2.
- SCTLRMASK_EL2.
- SCTLR2MASK_EL2.
- TCRMASK_EL2.
- TCR2MASK_EL2.
- ACTLRMASK_EL2, if it is implemented.

SRMASKEn	Meaning
0b0	EL2 accesses to the specified registers are trapped to EL3. EL1 accesses to the specified EL1 registers are trapped to EL3. The values in the registers are treated as 0.
0b1	No accesses are trapped by this control.

Traps are reported using EC syndrome value 0x18.

Traps generated by this control have a lower priority than traps generated by the HCRX_EL2.SRMASKEn, HFGRTR2_EL2 and HFGWTR2_EL2 controls.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PFAREn, bit [53]

When FEAT_PFAR is implemented:

Enable access to Physical Fault Address Registers. When disabled, accesses to Physical Fault Address Registers generate a trap to EL3.

PFAREn	Meaning
0b0	Accesses of the specified Physical Fault Address Registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	This control does not cause any instructions to be trapped.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to PFAR_EL1, PFAR_EL2, and PFAR_EL12.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWERR, bit [52]

When FEAT_RASv2 is implemented:

Trap writes of Error Record registers. Enables a trap to EL3 on writes of Error Record registers.

TWERR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Writes of the specified Error Record registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are: MSR accesses to ERRSELR_EL1, ERXADDR_EL1, ERXCTLR_EL1, ERXMISC0_EL1, ERXMISC1_EL1, ERXMISC2_EL1, ERXMISC3_EL1, and ERXSTATUS_EL1.

In AArch32 state, the instructions affected by this control are: MCR accesses to ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, ERXMISC4, ERXMISC5, ERXMISC6, ERXMISC7, and ERXSTATUS.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Trapped AArch32 instructions are reported using EC syndrome value 0x03.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMEA, bit [51]

When FEAT_DoubleFault2 is implemented:

Trap Masked External Aborts. Controls whether a masked error exception at a lower Exception level is taken to EL3.

TMEA	Meaning
0b0	Synchronous External abort exceptions and SError exceptions at EL2, EL1, and EL0 are unaffected by this mechanism. That is, these exceptions are not taken to EL3 unless routed to EL3 by another control.
0b1	When executing at Exception levels below EL3, all of the following apply: • When PSTATE.A is 1, synchronous External abort exceptions are taken to EL3, unless they are taken from EL1 or EL0 and routed to EL2 by another control. • Masked physical SError exceptions are taken to EL3, unless they are taken from EL1 or EL0 and routed to EL2 by another control.

This field has no effect on the routing of virtual or delegated SError exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [50]

When FEAT_FPMR is implemented:

Enables direct and indirect accesses to FPMR from EL2, EL1, and EL0.

When accesses to FPMR are disabled by this control:

- Direct accesses to FPMR from EL2, EL1, and EL0 are trapped to EL3 and reported with EC syndrome value `0x18`.
- Execution of FP8 data-processing instructions that indirectly access FPMR is UNDEFINED at EL2, EL1 and EL0.

EnFPM	Meaning
`0b0`	Direct and indirect accesses to FPMR are disabled at EL2, EL1 and EL0.
`0b1`	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

If EL3 is not implemented, the Effective value of this field is `0b1`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MECEn, bit [49]

When FEAT_MEC is implemented:

Enables access to the following EL2 MECID registers, from EL2:

- MECID_P0_EL2.
- MECID_A0_EL2
- MECID_P1_EL2
- MECID_A1_EL2
- VMECID_P_EL2
- VMECID_A_EL2

Accesses to these registers are trapped and reported using EC syndrome value `0x18`.

MECEn	Meaning
`0b0`	EL2 accesses to any of the specified registers are trapped to EL3. The values of the specified registers are treated as 0 for all purposes other than direct reads or writes to the register from EL3.
`0b1`	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

GPF, bit [48]

When FEAT_RME is implemented:

Controls the reporting of Granule protection faults at EL0, EL1 and EL2.

GPF	Meaning
0b0	This control does not cause exceptions to be routed from EL0, EL1 or EL2 to EL3.
0b1	GPFs at EL0, EL1 and EL2 are routed to EL3 and reported as Granule Protection Check exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D128En, bit [47]
When FEAT_D128 is implemented:

128-bit System Register trap control. Enables access to 128-bit System Registers via MRRS, MSRR instructions.

- MRRS and MSRR accesses from EL1 and EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value 0x14:
  - TTBR0_EL1.
  - TTBR1_EL1.
  - RCWMASK_EL1, RCWSMASK_EL1.
  - PAR_EL1.
- MRRS and MSRR accesses from EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value 0x14:
  - TTBR1_EL2 and accesses using the register name TTBR1_EL12.
  - TTBR0_EL2 and accesses using the register name TTBR0_EL12.
  - VTTBR_EL2.

D128En	Meaning
0b0	EL1 and EL2 accesses to the specified registers are disabled, and trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIEn, bit [46]
When FEAT_AIE is implemented:

MAIR2_ELx, AMAIR2_ELx Register access trap control.

- Accesses from EL1 and EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value 0x18:
  - AMAIR2_EL1.
  - MAIR2_EL1.
- Accesses from EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value 0x18:
  - AMAIR2_EL2 and accesses using the register name AMAIR2_EL12.
  - MAIR2_EL2 and accesses using the register name MAIR2_EL12.

AIEn	Meaning
0b0	EL1 and EL2 accesses to the specified registers are disabled, and trapped to EL3. The values in these registers are treated as 0.
0b1	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIEn, bit [45]

When FEAT_S1PIE is implemented, or FEAT_S2PIE is implemented, or FEAT_S1POE is implemented, or FEAT_S2POE is implemented:

Permission Indirection, Overlay Register access trap control. Enables access to Permission Indirection and Overlay registers.

- Accesses from EL0, EL1 and EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value 0x18:
  - POR_EL0.
- Accesses from EL1 and EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value 0x18:
  - PIRE0_EL1.
  - PIR_EL1.
  - POR_EL1.
  - S2POR_EL1.
- Accesses from EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value 0x18:
  - PIRE0_EL2 and accesses using the register name PIRE0_EL12.
  - PIR_EL2 and accesses using the register name PIR_EL12.
  - POR_EL2 and accesses using the register name POR_EL12.
  - S2PIR_EL2.

PIEn	Meaning
0b0	EL0, EL1 and EL2 accesses to the specified registers are disabled, and trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

If this field is 0, it is IMPLEMENTATION SPECIFIC whether the values of the named registers are treated as zero.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCTLR2En, bit [44]

When FEAT_SCTLR2 is implemented:

SCTLR2_ELx register trap control. Enables access to SCTLR2_EL1 and SCTLR2_EL2 registers.

SCTLR2En	Meaning
0b0	EL1 and EL2 accesses to SCTLR2_EL1 and SCTLR2_EL2 registers are disabled, and trapped to EL3. The values in these registers are treated as 0.
0b1	This control does not cause any instructions to be trapped.

Traps are reported using EC syndrome value 0x18.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCR2En, bit [43]

When FEAT_TCR2 is implemented:

TCR2_ELx register trap control. Enables access to TCR2_EL1 and TCR2_EL2 registers.

TCR2En	Meaning
0b0	EL1 and EL2 accesses to TCR2_EL1 and TCR2_EL2 registers are disabled, and trapped to EL3. The values in these registers are treated as 0.
0b1	This control does not cause any instructions to be trapped.

Traps are reported using EC syndrome value 0x18.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RCWMASKEn, bit [42]

When FEAT_THE is implemented:

RCW and RCWS Mask register trap control. Enables access to RCWMASK_EL1, RCWSMASK_EL1.

RCWMASKEn	Meaning
0b0	EL1 and EL2 accesses to RCWMASK_EL1 and RCWSMASK_EL1 registers are disabled, and trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

Traps for MRS, MSR access are reported using EC syndrome value 0x18.

Traps for MRRS, MSRR acceess are reported using EC syndrome value 0x14.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTP2, bit [41]
When FEAT_SME is implemented:

Traps instructions executed at EL2, EL1, and EL0 that access TPIDR2_EL0 to EL3. The exception is reported using EC syndrome value 0x18.

EnTP2	Meaning
0b0	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRNDR, bit [40]
When FEAT_RNG_TRAP is implemented:

Controls trapping of reads of RNDR and RNDRRS. The exception is reported using EC syndrome value 0x18.

TRNDR	Meaning
0b0	This control does not cause RNDR and RNDRRS to be trapped. When FEAT_RNG is implemented: • ID_AA64ISAR0_EL1.RNDR returns the value 0b0001. When FEAT_RNG is not implemented: • ID_AA64ISAR0_EL1.RNDR returns the value 0b0000. • MRS reads of RNDR and RNDRRS are UNDEFINED.
0b1	ID_AA64ISAR0_EL1.RNDR returns the value 0b0001. Any attempt to read RNDR or RNDRRS is trapped to EL3.

When FEAT_RNG is not implemented, Arm recommends that SCR_EL3.TRNDR is initialized before entering Exception levels below EL3 and not subsequently changed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

GCSEn, bit [39]
When FEAT_GCS is implemented:

Guarded Control Stack enable. Controls access to the Guarded Control Stack registers from EL2, EL1, and EL0, and controls whether the Guarded Control Stack is enabled.

The Guarded Control Stack registers trapped by this mechanism are:

- GCSCRE0_EL1.
- GCSCR_EL1.
- GCSCR_EL2.
- GCSCR_EL12.

- GCSPR_EL0.
- GCSPR_EL1.
- GCSPR_EL2.
- GCSPR_EL12.

GCSEn	Meaning
0b0	Trap read and write accesses to all Guarded Control Stack registers to EL3. All Guarded Control Stack behavior is disabled at EL2, EL1, and EL0.
0b1	This control does not cause any instructions to be trapped, and does not disable Guarded Control Stack behavior at EL2, EL1, or EL0.

Traps are reported using EC syndrome value 0x18.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HXEn, bit [38]
When FEAT_HCX is implemented:

Enables access to the HCRX_EL2 register at EL2 from EL3.

HXEn	Meaning
0b0	Accesses at EL2 to HCRX_EL2 are trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

When EL3 is not implemented, the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ADEn, bit [37]
When FEAT_LS64_ACCDATA is implemented:

Enables access to the ACCDATA_EL1 register at EL1 and EL2.

ADEn	Meaning
0b0	Accesses to ACCDATA_EL1 at EL1 and EL2 are trapped to EL3, unless the accesses are trapped to EL2 by the EL2 fine-grained trap.
0b1	This control does not cause accesses to ACCDATA_EL1 to be trapped.

If the HFGWTR_EL2.nACCDATA_EL1 or HFGRTR_EL2.nACCDATA_EL1 traps are enabled, they take priority over this trap.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [36]

When FEAT_LS64_ACCDATA is implemented:

Traps execution of an ST64BV0 instruction at EL0, EL1, or EL2 to EL3.

EnAS0	Meaning
0b0	EL0 execution of an ST64BV0 instruction is trapped to EL3, unless it is trapped to EL1 by SCTLR_EL1.EnAS0, or to EL2 by either HCRX_EL2.EnAS0 or SCTLR_EL2.EnAS0. EL1 execution of an ST64BV0 instruction is trapped to EL3, unless it is trapped to EL2 by HCRX_EL2.EnAS0. EL2 execution of an ST64BV0 instruction is trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV0 instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMVOFFEN, bit [35]

When FEAT_AMUv1p1 is implemented:

Activity Monitors Virtual Offsets Enable.

AMVOFFEN	Meaning
0b0	Accesses to AMEVCNTVOFF0<n>_EL2 and AMEVCNTVOFF1<n>_EL2 at EL2 are trapped to EL3. Indirect reads of the virtual offset registers are zero.
0b1	Accesses to AMEVCNTVOFF0<n>_EL2 and AMEVCNTVOFF1<n>_EL2 are not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME, bit [34]

When FEAT_TME is implemented:

Enables access to the TSTART, TCOMMIT, TTEST and TCANCEL instructions at EL0, EL1 and EL2.

TME	Meaning
0b0	EL0, EL1 and EL2 accesses to TSTART, TCOMMIT, TTEST and TCANCEL instructions are UNDEFINED.
0b1	This control does not cause any instruction to be UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEL, bits [33:30]
When FEAT_TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when SCR_EL3.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by SCR_EL3.TWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [29]
When FEAT_TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by SCR_EL3.TWE.

Traps are reported using EC syndrome value `0x01`.

TWEDEn	Meaning
`0b0`	The delay for taking the trap is IMPLEMENTATION DEFINED.
`0b1`	The delay for taking the trap is at least the number of cycles defined in SCR_EL3.TWEDEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECVEn, bit [28]
When FEAT_ECV_POFF is implemented:

ECV Enable. Enables access to the CNTPOFF_EL2 register.

ECVEn	Meaning
`0b0`	EL2 accesses to CNTPOFF_EL2 are trapped to EL3, and the value of CNTPOFF_EL2 is treated as 0 for all purposes other than direct reads or writes to the register from EL3.
`0b1`	EL2 accesses to CNTPOFF_EL2 are not trapped to EL3 by this mechanism.

When FEAT_ECV_POFF is not implemented, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FGTEn, bit [27]
When FEAT_FGT is implemented:

Fine-Grained Traps Enable. When EL2 is implemented, enables the traps to EL2 controlled by HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGRTR_EL2, HFGITR_EL2, and HFGWTR_EL2, and controls access to those registers.

Note

If EL2 is not implemented but EL3 is implemented, FEAT_FGT implements the MDCR_EL3.TDCC traps.

FGTEn	Meaning
0b0	EL2 accesses to HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGRTR_EL2, HFGITR_EL2 and HFGWTR_EL2 registers are trapped to EL3, and the traps to EL2 controlled by those registers are disabled.
0b1	EL2 accesses to HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGRTR_EL2, HFGITR_EL2 and HFGWTR_EL2 registers are not trapped to EL3 by this mechanism.

Traps caused by accesses to the fine-grained trap registers are reported using EC syndrome value 0x18 and its associated ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA, bit [26]
When FEAT_MTE2 is implemented:

Allocation Tag Access. Controls access to Allocation Tags, System registers for Memory tagging, and prevention of Tag checking, at EL2, EL1 and EL0.

ATA	Meaning
0b0	Access to Allocation Tags is prevented at EL2, EL1, and EL0. Accesses at EL1 and EL2 to GCR_EL1, RGSR_EL1, TFSR_EL1, TFSR_EL2 or TFSRE0_EL1 that are not UNDEFINED or trapped to a lower Exception level are trapped to EL3. Accesses at EL2 using MRS or MSR with the register name TFSR_EL12 that are not UNDEFINED are trapped to EL3. Memory accesses at EL2, EL1, and EL0 are not subject to a Tag Check operation.
0b1	This control does not prevent access to Allocation Tags at EL2, EL1, and EL0. This control does not prevent Tag checking at EL2, EL1, and EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSCXT, bit [25]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Enables access to the SCXTNUM_EL2, SCXTNUM_EL1, and SCXTNUM_EL0 registers.

EnSCXT	Meaning
0b0	Accesses at EL0, EL1 and EL2 to SCXTNUM_EL0, SCXTNUM_EL1, or SCXTNUM_EL2 registers are trapped to EL3 if they are not trapped by a higher priority exception, and the values of these registers are treated as 0.
0b1	This control does not cause any accesses to be trapped, or register values to be treated as 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [24]

Reserved, RES0.

TID5, bit [23]

When FEAT_IDTE3 is implemented and FEAT_MTE2 is implemented:

Trap ID group 5. EL2 and EL1 reads of the group 5 ID register GMID_EL1 are trapped to EL3, reported using EC syndrome value 0x18, unless the instruction generates a higher priority exception.

TID5	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL2 accesses to ID group 5 registers are trapped to EL3. When EL2 is not enabled in the current Security state, or when HCR_EL2.TID3 is 0, EL1 read accesses to the specified registers are trapped to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TID3, bit [22]

When FEAT_IDTE3 is implemented:

Trap ID group 3. EL2 and EL1 reads of the following group 3 registers are trapped to EL3, reported using EC syndrome value 0x18, unless the instruction generates a higher priority exception:

- ID_PFR0_EL1, ID_PFR1_EL1, ID_DFR0_EL1, ID_AFR0_EL1, ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, ID_ISAR5_EL1, MVFR0_EL1, MVFR1_EL1, and MVFR2_EL1.
- ID_AA64PFR0_EL1, ID_AA64PFR1_EL1, ID_AA64DFR0_EL1, ID_AA64DFR1_EL1, ID_AA64ISAR0_EL1, ID_AA64ISAR1_EL1, ID_AA64MMFR0_EL1, ID_AA64MMFR1_EL1, ID_AA64AFR0_EL1, and ID_AA64AFR1_EL1.
- ID_PFR2_EL1, ID_MMFR4_EL1 and ID_MMFR5_EL1.
- ID_AA64MMFR3_EL1.
- ID_AA64MMFR4_EL1.
- ID_AA64PFR2_EL1.
- ID_AA64MMFR2_EL1 and ID_ISAR6_EL1.
- ID_DFR1_EL1.

- ID_AA64ZFR0_EL1.
- ID_AA64SMFR0_EL1.
- ID_AA64ISAR2_EL1.
- This field traps all MRS accesses to registers in the following range that are not already mentioned in this field description: op0 == 3, op1 == 0, CRn == 0, CRm == {2-7}, op2 == {0-7}.

TID3	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL2 read accesses to ID group 3 registers are trapped to EL3. When EL2 is not enabled in the current Security state, or when HCR_EL2.TID3 is 0, EL1 read accesses to the specified registers are trapped to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FIEN, bit [21]
When FEAT_RASv1p1 is implemented:

Fault Injection enable. Trap accesses to the registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1 from EL1 and EL2 to EL3, reported using EC syndrome value 0x18.

FIEN	Meaning
0b0	Accesses to the specified registers from EL1 and EL2 generate a Trap exception to EL3.
0b1	This control does not cause any instructions to be trapped.

If EL3 is not implemented, the Effective value of SCR_EL3.FIEN is 0b1.

If ERRIDR_EL1.NUM is zero, meaning no error records are implemented, or no error record accessible using System registers is owned by a node that implements the RAS Common Fault Injection Model Extension, then this bit might be RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [20]
When FEAT_DoubleFault is implemented:

Non-maskable External Aborts. Controls whether PSTATE.A masks SError exceptions at EL3.

NMEA	Meaning
0b0	SError exceptions are not taken at EL3 if PSTATE.A == 1.
0b1	SError exceptions are taken at EL3 regardless of the value of PSTATE.A.

This field is ignored by the PE and treated as zero when all of the following are true:

- FEAT_DoubleFault2 is not implemented.
- SCR_EL3.EA is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

EASE, bit [19]
When FEAT_DoubleFault is implemented:

External aborts to SError exception vector.

EASE	Meaning
0b0	Synchronous External abort exceptions taken to EL3 are taken to the appropriate synchronous exception vector offset from VBAR_EL3.
0b1	Synchronous External abort exceptions taken to EL3 are taken to the appropriate SError exception vector offset from VBAR_EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

EEL2, bit [18]
When FEAT_SEL2 is implemented:

Secure EL2 Enable.

EEL2	Meaning
0b0	All behaviors associated with Secure EL2 are disabled. All registers, including timer registers, defined by FEAT_SEL2 are UNDEFINED, and those timers are disabled.
0b1	All behaviors associated with Secure EL2 are enabled.

When the value of this bit is 1, then:

- When SCR_EL3.NS == 0, the SCR_EL3.RW bit is treated as 1 for all purposes other than reading or writing the register.
- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2, using EC syndrome value 0x03 :
  - A read or write of the SCR.
  - A read or write of the NSACR.
  - A read or write of the MVBAR.
  - A read or write of the SDCR.
  - Execution of an ATS12NSO** instruction.
- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2 using EC syndrome value 0x00 :
  - Execution of an SRS instruction that uses R13_mon.
  - Execution of an MRS (Banked register) or MSR (Banked register) instruction that would access SPSR_mon, R13_mon, or R14_mon.

Note

If the Effective value of SCR_EL3.EEL2 is 0, then these operations executed in Secure EL1 using AArch32 are trapped to EL3.

A Secure only implementation that does not implement EL3 but implements EL2, behaves as if SCR_EL3.EEL2 == 1.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

API, bit [17]

When FEAT_PAuth is implemented:

Controls the use of the following instructions related to Pointer Authentication.

- PACGA.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB, RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETAA, ERETAB, LDRAA and LDRAB, when any of the following are true:
  - In EL0, when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and the associated SCTLR_EL1.En<N><M> == 1.
  - In EL0, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, and the associated SCTLR_EL2.En<N><M> == 1.
  - In EL1, when the associated SCTLR_EL1.En<N><M> == 1.
  - In EL2, when the associated SCTLR_EL2.En<N><M> == 1.
- When FEAT_PAuth_LR is implemented, AUTIASPPC, AUTIASPPCR, AUTIA171615, AUTIBSPPC, AUTIBSPPCR, AUTIB171615, PACIASPPC, PACNBIASPPC, PACIA171615, PACIBSPPC, PACNBIBSPPC, PACIB171615, RETAASPPC, RETAASPPCR, RETABSPPC, RETABSPPCR, when any of the following are true:
  - In EL0, when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and the associated SCTLR_EL1.En<N><M> == 1.
  - In EL0, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, and the associated SCTLR_EL2.En<N><M> == 1.
  - In EL1, when the associated SCTLR_EL1.En<N><M> == 1.
  - In EL2, when the associated SCTLR_EL2.En<N><M> == 1.

API	Meaning
0b0	The specified instructions are trapped to EL3, when the instructions are enabled, unless they are trapped to EL2 as a result of the higher priority HCR_EL2.API trap.
0b1	This control does not cause any instructions to be trapped.

Traps are reported using EC syndrome value 0x09.

An instruction is trapped only if Pointer Authentication is enabled for that instruction, for more information, see 'PAC generation and verification keys'.

Note

If FEAT_PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APK, bit [16]

When FEAT_PAuth is implemented:

Trap registers holding "key" values for Pointer Authentication. Traps accesses to the following registers, using EC syndrome value 0x18, from EL1 or EL2 to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.APK bit or other traps:

- APIAKeyLo_EL1, APIAKeyHi_EL1, APIBKeyLo_EL1, APIBKeyHi_EL1.
- APDAKeyLo_EL1, APDAKeyHi_EL1, APDBKeyLo_EL1, APDBKeyHi_EL1.
- APGAKeyLo_EL1, and APGAKeyHi_EL1.

APK	Meaning
0b0	Access to the registers holding "key" values for pointer authentication from EL1 or EL2 are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.APK bit or other traps.
0b1	This control does not cause any instructions to be trapped.

For more information, see 'PAC generation and verification keys'.

Note

If FEAT_PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TERR, bit [15]

When FEAT_RAS is implemented:

Trap accesses of Error Record registers. Enables a trap to EL3 on accesses of Error Record registers.

TERR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Accesses of the specified Error Record registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to ERRSELR_EL1, ERXADDR_EL1, ERXCTLR_EL1, ERXMISC0_EL1, ERXMISC1_EL1, and ERXSTATUS_EL1.
- MRS accesses to ERRIDR_EL1 and ERXFR_EL1.
- If FEAT_RASv1p1 is implemented, MRS and MSR accesses to ERXMISC2_EL1 and ERXMISC3_EL1.
- If FEAT_RASv2 is implemented, MRS accesses to ERXGSR_EL1.

In AArch32 state, the instructions affected by this control are:

- MRC and MCR accesses to ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS.
- MRC accesses to ERRIDR, ERXFR, and ERXFR2.
- If FEAT_RASv1p1 is implemented, MRC and MCR accesses to ERXMISC4, ERXMISC5, ERXMISC6, and ERXMISC7.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Trapped AArch32 instructions are reported using EC syndrome value 0x03.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLOR, bit [14]

When FEAT_LOR is implemented:

Trap LOR registers. Traps Non-secure and Realm accesses to the LORSA_EL1, LOREA_EL1, LORN_EL1, LORC_EL1, and LORID_EL1 registers from EL1 and EL2 to EL3, unless the access has been trapped to EL2.

TLOR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure and Realm EL1 and EL2 accesses to the LOR registers that are not UNDEFINED are trapped to EL3, unless it is trapped by HCR_EL2.TLOR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWE, bit [13]

Traps EL2, EL1, and EL0 execution of WFE instructions to EL3, from any Security state and both Execution states, reported using EC syndrome value 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

TWE	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFE instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE, HCR.TWE, SCTLR_EL1.nTWE, SCTLR_EL2.nTWE, or HCR_EL2.TWE.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

For more information about when WFE instructions can cause the PE to enter a low-power state, see 'Wait for Event mechanism and Send event'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TWI, bit [12]

Traps EL2, EL1, and EL0 execution of WFI instructions to EL3, from any Security state and both Execution states, reported using EC syndrome value 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

TWI	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFI instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI, HCR.TWI, SCTLR_EL1.nTWI, SCTLR_EL2.nTWI, or HCR_EL2.TWI.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

For more information about when WFI instructions can cause the PE to enter a low-power state, see 'Wait for Interrupt'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ST, bit [11]

Traps Secure EL1 accesses to the Counter-timer Physical Secure timer registers to EL3, from AArch64 state only, reported using EC syndrome value 0x18.

ST	Meaning
0b0	Secure EL1 using AArch64 accesses to the CNTPS_TVAL_EL1, CNTPS_CTL_EL1, and CNTPS_CVAL_EL1 are trapped to EL3 when Secure EL2 is disabled. If Secure EL2 is enabled, the behavior is as if the value of this field was 0b1.
0b1	This control does not cause any instructions to be trapped.

Note

Accesses to the Counter-timer Physical Secure timer registers are always enabled at EL3. These registers are not accessible at EL0.

When FEAT_RME is implemented and Secure state is not implemented, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RW, bit [10]

When FEAT_AA32EL1 is implemented:

Execution state control for lower Exception levels.

RW	Meaning
0b0	All lower Exception levels are using AArch32.
0b1	The next lower Exception level is using AArch64: • If EL2 is implemented and enabled in the Security state determined by SCR_EL3.{NSE,NS}, then EL2 is using AArch64 and EL2 controls the Execution state for EL1. • If EL2 is not implemented or EL2 is disabled in the Security state determined by SCR_EL3.{NSE,NS}, then EL1 is using AArch64. When executing at EL0, if the next higher Exception level is using AArch64, then the Execution state is determined by PSTATE.nRW. Otherwise, EL0 uses AArch32.

If any of the following apply, then the Effective value of this bit is 1:

- EL2 is implemented and does not support AArch32, and SCR_EL3.NS is 1.

- The Effective value of SCR_EL3.{EEL2,NS} is {1,0}.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAO/WI.

SIF, bit [9]

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction execution from memory marked in the first stage of translation as being Non-secure.

SIF	Meaning
0b0	Secure state instruction execution from memory marked in the first stage of translation as being Non-secure is permitted.
0b1	Secure state instruction execution from memory marked in the first stage of translation as being Non-secure is not permitted.

When FEAT_RME is implemented and Secure state is not implemented, this bit is RES0.

When FEAT_PAN3 is implemented, it is IMPLEMENTATION DEFINED whether SCR_EL3.SIF is also used to determine instruction access permission for the purpose of PAN.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HCE, bit [8]

Hypervisor Call instruction enable. Enables HVC instructions at EL3 and, if EL2 is enabled in the current Security state, at EL2 and EL1, in both Execution states, reported using EC syndrome value 0x00.

HCE	Meaning
0b0	HVC instructions are UNDEFINED.
0b1	HVC instructions are enabled at EL3, EL2, and EL1.

Note

HVC instructions are always UNDEFINED at EL0 and, if Secure EL2 is disabled, at Secure EL1. Any resulting exception is taken from the current Exception level to the current Exception level.

If EL2 is not implemented, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SMD, bit [7]

Secure Monitor Call disable. Disables SMC instructions at EL1 and above, from any Security state and both Execution states, reported using EC syndrome value 0x00.

SMD	Meaning
0b0	SMC instructions are enabled at EL3, EL2 and EL1.
0b1	SMC instructions are UNDEFINED.

Note

SMC instructions are always UNDEFINED at EL0. Any resulting exception is taken from the current Exception level to the current Exception level.

If HCR_EL2.TSC or HCR.TSC traps attempted EL1 execution of SMC instructions to EL2, that trap has priority over this disable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

Bits [5:4]

Reserved, RES1.

EA, bit [3]

External Abort and SError exception routing.

EA	Meaning
0b0	When executing at Exception levels below EL3, External aborts and SError exceptions are not taken to EL3. In addition, when executing at EL3: • SError exceptions are not taken. • External aborts are taken to EL3.
0b1	When executing at any Exception level, External aborts and SError exceptions are taken to EL3.

This field has no effect on the routing of virtual or delegated SError exceptions.

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FIQ, bit [2]

Physical FIQ Routing.

FIQ	Meaning
0b0	When executing at Exception levels below EL3, physical FIQ interrupts are not taken to EL3. When executing at EL3, physical FIQ interrupts are not taken.
0b1	When executing at any Exception level, physical FIQ interrupts are taken to EL3.

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRQ, bit [1]

Physical IRQ Routing.

IRQ	Meaning
0b0	When executing at Exception levels below EL3, physical IRQ interrupts are not taken to EL3. When executing at EL3, physical IRQ interrupts are not taken.
0b1	When executing at any Exception level, physical IRQ interrupts are taken to EL3.

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [0]

When FEAT_RME is implemented:

Non-secure bit. This field is used in combination with SCR_EL3.NSE to select the Security state of EL2 and lower Exception levels.

NSE	NS	Meaning
0b0	0b0	Secure.
0b0	0b1	Non-secure.
0b1	0b0	Reserved.
0b1	0b1	Realm.

When Secure state is not implemented, SCR_EL3.NS is RES1 and its effective value is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure bit.

NS	Meaning
0b0	Indicates that EL0 and EL1 are in Secure state. When FEAT_SEL2 is implemented and SCR_EL3.EEL2 == 1, then EL2 is using AArch64 and in Secure state.
0b1	Indicates that Exception levels lower than EL3 are in Non-secure state, so memory accesses from those Exception levels cannot access Secure memory.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCR_EL3;
```

```
MSR SCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR2_EL1, System Control Register (EL1)

The SCTLR2_EL1 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL1 and EL0.

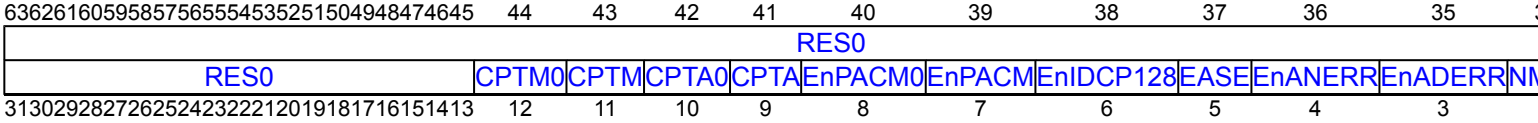
Configuration

This register is present only when FEAT_SCTLR2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR2_EL1 are UNDEFINED.

Attributes

SCTLR2_EL1 is a 64-bit register.

Field descriptions



Bits [63:13]

Reserved, RES0.

CPTM0, bit [12]

When FEAT_CPA2 is implemented:

This field controls unprivileged Checked Pointer Arithmetic for Multiplication.

CPTM0	Meaning
0b0	Pointer Arithmetic for Multiplication is not checked.
0b1	Pointer Arithmetic for Multiplication is checked.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

If the Effective value of SCTLR2_EL1.CPTA0 is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTM, bit [11]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Multiplication at EL1.

CPTM	Meaning
0b0	Pointer Arithmetic for Multiplication is not checked.
0b1	Pointer Arithmetic for Multiplication is checked.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

If the Effective value of SCTLR2_EL1.CPTA is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA0, bit [10]

When FEAT_CPA2 is implemented:

This field controls unprivileged Checked Pointer Arithmetic for Addition.

CPTA0	Meaning
0b0	Pointer Arithmetic for Addition is not checked.
0b1	Pointer Arithmetic for Addition is checked.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA, bit [9]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Addition at EL1.

CPTA	Meaning
0b0	Pointer Arithmetic for Addition is not checked.
0b1	Pointer Arithmetic for Addition is checked.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM0, bit [8]
When FEAT_PAuth_LR is implemented:

PACM Enable at EL0. Controls the effect of a PACM instruction at EL0.

EnPACM0	Meaning
0b0	The effects of PACM are disabled at EL0.
0b1	A PACM instruction at EL0 causes PSTATE.PACM to be set to 0b1.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.PACMEn == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM, bit [7]
When FEAT_PAuth_LR is implemented:

PACM Enable at EL1. Controls the effect of a PACM instruction at EL1.

EnPACM	Meaning
0b0	The effects of PACM are disabled at EL1.
0b1	A PACM instruction at EL1 causes PSTATE.PACM to be set to 0b1.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.PACMEn == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIDCP128, bit [6]

When FEAT_SYSREG128 is implemented:

Enables access to IMPLEMENTATION DEFINED 128-bit System registers.

EnIDCP128	Meaning
0b0	Accesses at EL0 to IMPLEMENTATION DEFINED 128-bit System registers are trapped to EL1 using an ESR_EL1.EC value of 0x14, unless the access generates a higher priority exception. Disables the functionality of the 128-bit IMPLEMENTATION DEFINED System registers that are accessible at EL1.
0b1	No accesses are trapped by this control.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EASE, bit [5]

When FEAT_DoubleFault2 is implemented:

External Aborts to SError exception vector.

EASE	Meaning
0b0	Synchronous External abort exceptions taken to EL1 are taken to the appropriate synchronous exception vector offset from VBAR_EL1.
0b1	Synchronous External abort exceptions taken to EL1 are taken to the appropriate SError exception vector offset from VBAR_EL1.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnANERR, bit [4]

When FEAT_ANERR is implemented:

Enable Asynchronous Normal Read Error.

EnANERR	Meaning
0b0	External aborts on Normal memory reads generate synchronous Data Abort exceptions in the EL1&0 translation regime.
0b1	External aborts on Normal memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL1&0 translation regime.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Normal memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL1.EnANERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Normal memory reads.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.EnSNERR is 1.

Otherwise, this field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ADERR is implemented.
- ID_AA64MMFR3_EL1.ANERR reads as 0b0010.
- SCTLR2_EL1.EnADERR is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnADERR, bit [3]

When FEAT_ADERR is implemented:

Enable Asynchronous Device Read Error.

EnADERR	Meaning
0b0	External aborts on Device memory reads generate synchronous Data Abort exceptions in the EL1&0 translation regime.
0b1	External aborts on Device memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL1&0 translation regime.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Device memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL1.EnADERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Device memory reads.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.EnSDERR is 1.

Otherwise, this field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ANERR is implemented.
- ID_AA64MMFR3_EL1.ADERR reads as `0b0010`.
- SCTLR2_EL1.EnANERR is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [2]
When FEAT_DoubleFault2 is implemented:

Non-maskable External Aborts. Controls whether PSTATE.A masks SError exceptions at EL1.

NMEA	Meaning
`0b0`	SError exceptions are not taken at EL1 if PSTATE.A == 1, unless routed to a higher Exception level.
`0b1`	SError exceptions are taken at EL1 regardless of the value of PSTATE.A, unless routed to a higher Exception level.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [1:0]

Reserved, RES0.

Accessing SCTLR2_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name SCTLR2_EL1 or SCTLR2_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to SCTLR2_EL1 are masked by SCTLR2MASK_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x278];
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLR2_EL2;
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2_EL1;
```

```
MSR SCTLR2_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x278] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL1 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL1()) OR (SCTLR2_EL1 AND
EffectiveSCTLR2MASK_EL1());
        else
            SCTLR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL2 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL2()) OR (SCTLR2_EL2 AND
EffectiveSCTLR2MASK_EL2());
        else
            SCTLR2_EL2 = X[t, 64];
    else
        SCTLR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, SCTLR2_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x278];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SCTLR2_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLR2_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR SCTLR2_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x278] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SCTLR2_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SCTLR2_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_SRMASK is implemented

```
MRS <Xt>, SCTLR2ALIAS_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nSCTLRALIAS2_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x278];
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLR2_EL2;
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2_EL1;
```

When FEAT_SRMASK is implemented

```
MSR SCTLR2ALIAS_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nSCTLRALIAS2_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x278] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL1 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL1()) OR (SCTLR2_EL1 AND
EffectiveSCTLR2MASK_EL1());
        else
            SCTLR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL2 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL2()) OR (SCTLR2_EL2 AND
EffectiveSCTLR2MASK_EL2());
        else
            SCTLR2_EL2 = X[t, 64];
    else
        SCTLR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR2_EL2, System Control Register (EL2)

The SCTLR2_EL2 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL2.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, these controls also apply to execution at EL0.

Configuration

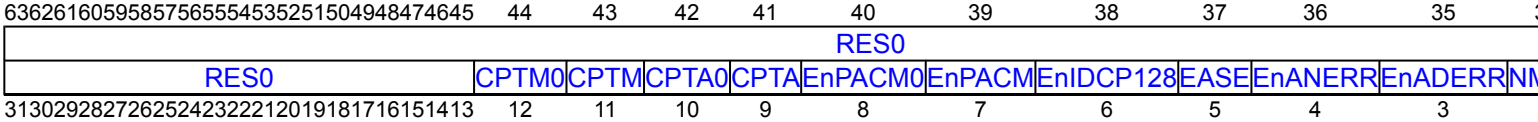
This register is present only when FEAT_SCTLR2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR2_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

SCTLR2_EL2 is a 64-bit register.

Field descriptions



Bits [63:13]

Reserved, RES0.

CPTM0, bit [12]

When FEAT_CPA2 is implemented and ELIsInHost(EL2):

This field controls unprivileged Checked Pointer Arithmetic for Multiplication.

CPTM0	Meaning
0b0	Pointer Arithmetic for Multiplication is not checked.
0b1	Pointer Arithmetic for Multiplication is checked.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

If the Effective value of SCTLR2_EL2.CPTA0 is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTM, bit [11]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Multiplication at EL2.

CPTM	Meaning
0b0	Pointer Arithmetic for Multiplication is not checked.
0b1	Pointer Arithmetic for Multiplication is checked.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

If the Effective value of SCTLR2_EL2.CPTA is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA0, bit [10]

When FEAT_CPA2 is implemented and ELIsInHost(EL2):

This field controls unprivileged Checked Pointer Arithmetic for Addition.

CPTA0	Meaning
0b0	Pointer Arithmetic for Addition is not checked.
0b1	Pointer Arithmetic for Addition is checked.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA, bit [9]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Addition at EL2.

CPTA	Meaning
0b0	Pointer Arithmetic for Addition is not checked.
0b1	Pointer Arithmetic for Addition is checked.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM0, bit [8]
When FEAT_PAuth_LR is implemented and ELIsInHost(EL2):

PACM Enable at EL0. Controls the effect of a PACM instruction at EL0.

EnPACM0	Meaning
0b0	The effects of PACM are disabled at EL0.
0b1	A PACM instruction at EL0 causes PSTATE.PACM to be set to 0b1.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM, bit [7]
When FEAT_PAuth_LR is implemented:

PACM Enable at EL2. Controls the effect of a PACM instruction at EL2.

EnPACM	Meaning
0b0	The effects of PACM are disabled at EL2.
0b1	A PACM instruction at EL2 causes PSTATE.PACM to be set to 0b1.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIDCP128, bit [6]
When FEAT_SYSREG128 is implemented:

Enables access to IMPLEMENTATION DEFINED 128-bit System registers.

EnIDCP128	Meaning
0b0	Accesses at EL0 to IMPLEMENTATION DEFINED 128-bit System registers are trapped to EL2 using an ESR_EL2.EC value of 0x14, unless the access generates a higher priority exception. Disables the functionality of the 128-bit IMPLEMENTATION DEFINED System registers that are accessible at EL2.
0b1	No accesses are trapped by this control.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EASE, bit [5]

When FEAT_DoubleFault2 is implemented:

External Aborts to SError exception vector.

EASE	Meaning
0b0	Synchronous External abort exceptions taken to EL2 are taken to the appropriate synchronous exception vector offset from VBAR_EL2.
0b1	Synchronous External abort exceptions taken to EL2 are taken to the appropriate SError exception vector offset from VBAR_EL2.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnANERR, bit [4]

When FEAT_ANERR is implemented:

Enable Asynchronous Normal Read Error.

EnANERR	Meaning
0b0	External aborts on Normal memory reads generate synchronous Data Abort exceptions in the EL2 and EL2&0 translation regimes.
0b1	External aborts on Normal memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL2 and EL2&0 translation regimes.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Normal memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL2.EnANERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Normal memory reads.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

Otherwise, this field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ADERR is implemented.
- ID_AA64MMFR3_EL1.ANERR reads as `0b0010`.
- SCTLR2_EL2.EnADERR is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnADERR, bit [3]
When FEAT_ADERR is implemented:

Enable Asynchronous Device Read Error.

EnADERR	Meaning
`0b0`	External aborts on Device memory reads generate synchronous Data Abort exceptions in the EL2 and EL2&0 translation regimes.
`0b1`	External aborts on Device memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL2 and EL2&0 translation regimes.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Device memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL2.EnADERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Device memory reads.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

Otherwise, this field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ANERR is implemented.
- ID_AA64MMFR3_EL1.ADERR reads as `0b0010`.
- SCTLR2_EL2.EnANERR is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [2]

When FEAT_DoubleFault2 is implemented:

Non-maskable External Aborts. Controls whether PSTATE.A masks SError exceptions at EL2.

NMEA	Meaning
0b0	SError exceptions are not taken at EL2 if PSTATE.A == 1, unless routed to a higher Exception level.
0b1	SError exceptions are taken at EL2 regardless of the value of PSTATE.A, unless routed to a higher Exception level.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EMEC, bit [1]

When FEAT_MEC is implemented:

Enables MEC. When enabled, memory accesses to the Realm physical address space are associated with a MECID.

EMEC	Meaning
0b0	MEC is not enabled for the Realm physical address space.
0b1	MEC is enabled for the Realm physical address space.

This bit is permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [0]

Reserved, RES0.

Accessing SCTLR2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SCTLR2_EL2 or SCTLR2_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to SCTLR2_EL2 are masked by SCTLR2MASK_EL2.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SCTLR2_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCTLR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2_EL2;
```

MSR SCTLR2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL2 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL2()) OR (SCTLR2_EL2 AND
EffectiveSCTLR2MASK_EL2());
        else
            SCTLR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2_EL2 = X[t, 64];
```

MRS <Xt>, SCTLR2_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x278];
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLR2_EL2;
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2_EL1;
```

```
MSR SCTLR2_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x278] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL1 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL1()) OR (SCTLR2_EL1 AND
EffectiveSCTLR2MASK_EL1());
        else
            SCTLR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL2 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL2()) OR (SCTLR2_EL2 AND
EffectiveSCTLR2MASK_EL2());
        else
            SCTLR2_EL2 = X[t, 64];
    else
        SCTLR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR2_EL3, System Control Register (EL3)

The SCTLR2_EL3 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL3.

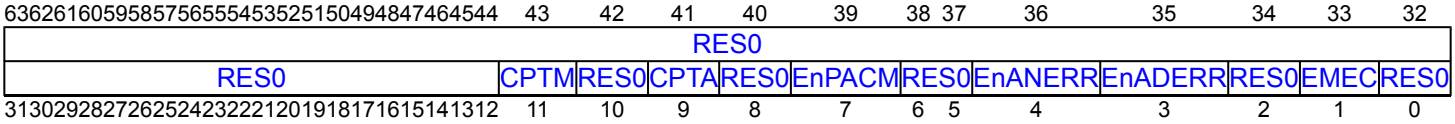
Configuration

This register is present only when FEAT_SCTLR2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR2_EL3 are UNDEFINED.

Attributes

SCTLR2_EL3 is a 64-bit register.

Field descriptions



Bits [63:12]

Reserved, RES0.

CPTM, bit [11]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Multiplication at EL3.

CPTM	Meaning
0b0	Pointer Arithmetic for Multiplication is not checked.
0b1	Pointer Arithmetic for Multiplication is checked.

If the Effective value of SCTLR2_EL3.CPTA is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [10]

Reserved, RES0.

CPTA, bit [9]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Addition at EL3.

CPTA	Meaning
0b0	Pointer Arithmetic for Addition is not checked.
0b1	Pointer Arithmetic for Addition is checked.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

EnPACM, bit [7]

When FEAT_PAuth_LR is implemented:

PACM Enable at EL3. Controls the effect of a PACM instruction at EL3.

EnPACM	Meaning
0b0	The effects of PACM are disabled at EL3.
0b1	A PACM instruction at EL3 causes PSTATE.PACM to be set to 0b1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EnANERR, bit [4]

When FEAT_ANERR is implemented:

Enable Asynchronous Normal Read Error.

EnANERR	Meaning
0b0	External aborts on Normal memory reads generate synchronous Data Abort exceptions in the EL3 translation regime.
0b1	External aborts on Normal memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL3 translation regime.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Normal memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL3.EnANERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Normal memory reads.

This field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ADERR is implemented.
- ID_AA64MMFR3_EL1.ANERR reads as `0b0010`.
- SCTLR2_EL3.EnADERR is 1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

EnADERR, bit [3]

When FEAT_ADERR is implemented:

Enable Asynchronous Device Read Error.

EnADERR	Meaning
`0b0`	External aborts on Device memory reads generate synchronous Data Abort exceptions in the EL3 translation regime.
`0b1`	External aborts on Device memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL3 translation regime.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Device memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL3.EnADERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Device memory reads.

This field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ANERR is implemented.
- ID_AA64MMFR3_EL1.ADERR reads as `0b0010`.
- SCTLR2_EL3.EnANERR is 1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

Bit [2]

Reserved, RES0.

EMEC, bit [1]

When FEAT_MEC is implemented:

Enables MEC. When enabled, memory accesses to the Realm physical address space are associated with MECID_RL_A_EL3.

EMEC	Meaning
0b0	MEC is not enabled for the Realm physical address space.
0b1	MEC is enabled for the Realm physical address space.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [0]

Reserved, RES0.

Accessing SCTLR2_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLR2_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2_EL3;
```

```
MSR SCTLR2_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.SCTLR2_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCTLR2_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR2MASK_EL1, Extended System Control Masking Register (EL1)

The SCTLR2MASK_EL1 characteristics are:

Purpose

Mask register to prevent updates of fields in SCTLR2_EL1 on writes to SCTLR2_EL1 or SCTLR2ALIAS_EL1.

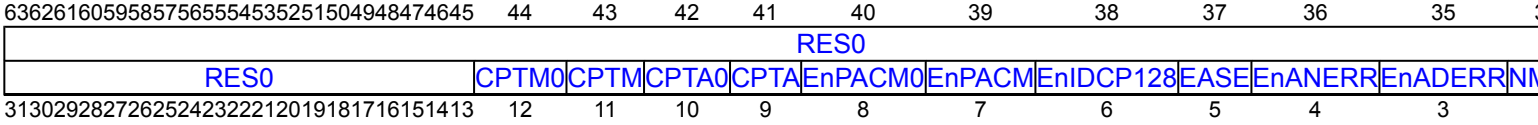
Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR2MASK_EL1 are UNDEFINED.

Attributes

SCTLR2MASK_EL1 is a 64-bit register.

Field descriptions



Bits [63:13]

Reserved, RES0.

CPTM0, bit [12]

When FEAT_CPA2 is implemented:

Mask bit for CPTM0.

CPTM0	Meaning
0b0	SCTLR2_EL1.CPTM0 is writeable.
0b1	SCTLR2_EL1.CPTM0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTM, bit [11]

When FEAT_CPA2 is implemented:

Mask bit for CPTM.

CPTM	Meaning
0b0	SCTLR2_EL1.CPTM is writeable.
0b1	SCTLR2_EL1.CPTM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA0, bit [10]
When FEAT_CPA2 is implemented:

Mask bit for CPTA0.

CPTA0	Meaning
0b0	SCTLR2_EL1.CPTA0 is writeable.
0b1	SCTLR2_EL1.CPTA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA, bit [9]
When FEAT_CPA2 is implemented:

Mask bit for CPTA.

CPTA	Meaning
0b0	SCTLR2_EL1.CPTA is writeable.
0b1	SCTLR2_EL1.CPTA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM0, bit [8]
When FEAT_PAuth_LR is implemented:

Mask bit for EnPACM0.

EnPACM0	Meaning
0b0	SCTLR2_EL1.EnPACM0 is writeable.
0b1	SCTLR2_EL1.EnPACM0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM, bit [7]
When FEAT_PAuth_LR is implemented:

Mask bit for EnPACM.

EnPACM	Meaning
0b0	SCTLR2_EL1.EnPACM is writeable.
0b1	SCTLR2_EL1.EnPACM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIDCP128, bit [6]
When FEAT_SYSREG128 is implemented:

Mask bit for EnIDCP128.

EnIDCP128	Meaning
0b0	SCTLR2_EL1.EnIDCP128 is writeable.
0b1	SCTLR2_EL1.EnIDCP128 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EASE, bit [5]
When FEAT_DoubleFault2 is implemented:

Mask bit for EASE.

EASE	Meaning
0b0	SCTLR2_EL1.EASE is writeable.
0b1	SCTLR2_EL1.EASE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnANERR, bit [4]

When FEAT_ANERR is implemented:

Mask bit for EnANERR.

EnANERR	Meaning
0b0	SCTLR2_EL1.EnANERR is writeable.
0b1	SCTLR2_EL1.EnANERR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnADERR, bit [3]

When FEAT_ADERR is implemented:

Mask bit for EnADERR.

EnADERR	Meaning
0b0	SCTLR2_EL1.EnADERR is writeable.
0b1	SCTLR2_EL1.EnADERR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [2]

When FEAT_DoubleFault2 is implemented:

Mask bit for NMEA.

NMEA	Meaning
0b0	SCTLR2_EL1.NMEA is writeable.
0b1	SCTLR2_EL1.NMEA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [1:0]

Reserved, RES0.

Accessing SCTLR2MASK_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name SCTLR2MASK_EL1 or SCTLR2MASK_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLR2MASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nSCTLR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x328];
    else
        X[t, 64] = SCTLR2MASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLR2MASK_EL2;
    else
        X[t, 64] = SCTLR2MASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2MASK_EL1;
```

```
MSR SCTLR2MASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nSCTLR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x328] = X[t, 64];
    elsif !IsZero(EffectiveSCTLR2MASK_EL1()) then
        UNDEFINED;
    else
        SCTLR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveSCTLR2MASK_EL2()) then
            UNDEFINED;
        else
            SCTLR2MASK_EL2 = X[t, 64];
    else
        SCTLR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2MASK_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, SCTLR2MASK_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x328];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SCTLR2MASK_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLR2MASK_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR SCTLR2MASK_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x328] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SCTLR2MASK_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SCTLR2MASK_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR2MASK_EL2, Extended System Control Masking Register (EL2)

The SCTLR2MASK_EL2 characteristics are:

Purpose

Mask register to prevent updates of fields in SCTLR2_EL2 on writes.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR2MASK_EL2 are UNDEFINED.

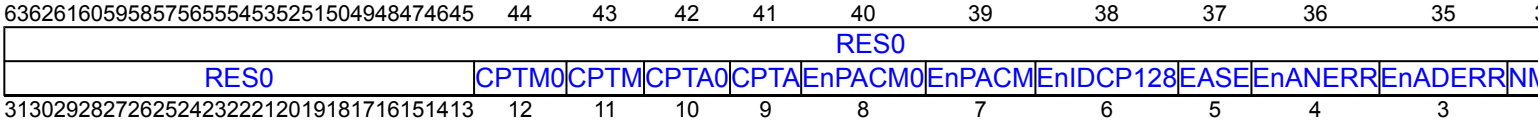
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

SCTLR2MASK_EL2 is a 64-bit register.

Field descriptions



Bits [63:13]

Reserved, RES0.

CPTM0, bit [12]

When FEAT_CPA2 is implemented:

Mask bit for CPTM0.

CPTM0	Meaning
0b0	SCTLR2_EL2.CPTM0 is writeable.
0b1	SCTLR2_EL2.CPTM0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTM, bit [11]

When FEAT_CPA2 is implemented:

Mask bit for CPTM.

CPTM	Meaning
0b0	SCTLR2_EL2.CPTM is writeable.
0b1	SCTLR2_EL2.CPTM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA0, bit [10]

When FEAT_CPA2 is implemented:

Mask bit for CPTA0.

CPTA0	Meaning
0b0	SCTLR2_EL2.CPTA0 is writeable.
0b1	SCTLR2_EL2.CPTA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA, bit [9]

When FEAT_CPA2 is implemented:

Mask bit for CPTA.

CPTA	Meaning
0b0	SCTLR2_EL2.CPTA is writeable.
0b1	SCTLR2_EL2.CPTA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM0, bit [8]

When FEAT_PAuth_LR is implemented:

Mask bit for EnPACM0.

EnPACM0	Meaning
0b0	SCTLR2_EL2.EnPACM0 is writeable.
0b1	SCTLR2_EL2.EnPACM0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM, bit [7]

When FEAT_PAuth_LR is implemented:

Mask bit for EnPACM.

EnPACM	Meaning
0b0	SCTLR2_EL2.EnPACM is writeable.
0b1	SCTLR2_EL2.EnPACM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIDCP128, bit [6]

When FEAT_SYSREG128 is implemented:

Mask bit for EnIDCP128.

EnIDCP128	Meaning
0b0	SCTLR2_EL2.EnIDCP128 is writeable.
0b1	SCTLR2_EL2.EnIDCP128 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EASE, bit [5]

When FEAT_DoubleFault2 is implemented:

Mask bit for EASE.

EASE	Meaning
0b0	SCTLR2_EL2.EASE is writeable.
0b1	SCTLR2_EL2.EASE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnANERR, bit [4]

When FEAT_ANERR is implemented:

Mask bit for EnANERR.

EnANERR	Meaning
0b0	SCTLR2_EL2.EnANERR is writeable.
0b1	SCTLR2_EL2.EnANERR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnADERR, bit [3]

When FEAT_ADERR is implemented:

Mask bit for EnADERR.

EnADERR	Meaning
0b0	SCTLR2_EL2.EnADERR is writeable.
0b1	SCTLR2_EL2.EnADERR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [2]

When FEAT_DoubleFault2 is implemented:

Mask bit for NMEA.

NMEA	Meaning
0b0	SCTLR2_EL2.NMEA is writeable.
0b1	SCTLR2_EL2.NMEA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EMEC, bit [1]

When FEAT_MEC is implemented:

Mask bit for EMEC.

EMEC	Meaning
0b0	SCTLR2_EL2.EMEC is writeable.
0b1	SCTLR2_EL2.EMEC is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [0]

Reserved, RES0.

Accessing SCTLR2MASK_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SCTLR2MASK_EL2 or SCTLR2MASK_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLR2MASK_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCTLR2MASK_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2MASK_EL2;
```

```
MSR SCTLR2MASK_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsZero(EffectiveSCTLR2MASK_EL2()) then
        UNDEFINED;
    else
        SCTLR2MASK_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2MASK_EL2 = X[t, 64];
```

```
MRS <Xt>, SCTLR2MASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nSCTLR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x328];
    else
        X[t, 64] = SCTLR2MASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLR2MASK_EL2;
    else
        X[t, 64] = SCTLR2MASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2MASK_EL1;
```

```
MSR SCTLR2MASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nSCTLR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x328] = X[t, 64];
    elsif !IsZero(EffectiveSCTLR2MASK_EL1()) then
        UNDEFINED;
    else
        SCTLR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveSCTLR2MASK_EL2()) then
            UNDEFINED;
        else
            SCTLR2MASK_EL2 = X[t, 64];
    else
        SCTLR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2MASK_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR_EL1, System Control Register (EL1)

The SCTLR_EL1 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL1 and EL0.

Configuration

AArch64 System register SCTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register SCTLR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR_EL1 are UNDEFINED.

Attributes

SCTLR_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	
TIDCP	SPINTMASK	NMI	EnTP2	TCSO	TCSO0	EPAN	EnALS	EnAS0	EnASR	TME	TME0	TMT	TMT0	TWEDEL				TW
EnIA	EnIB	LSMAOE	nTLSMD	EnDA	UCI	EE	E0E	SPAN	EIS	IESB	TSCXT	WXN	nTWE	RES0	nTWI	UCT	DZE	E
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	

TIDCP, bit [63]

When FEAT_TIDCP1 is implemented:

Trap IMPLEMENTATION DEFINED functionality. When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps EL0 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL1.

TIDCP	Meaning
0b0	No instructions accessing the System register or System instruction spaces are trapped by this mechanism.
0b1	Instructions accessing the following System register or System instruction spaces are trapped to EL1 by this mechanism: • In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped: ◦ IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}, and are reported using EC syndrome value 0x18. ◦ IMPLEMENTATION DEFINED System instructions, which are accessed using SYSP, with CRn == {11, 15}, and are reported using EC syndrome value 0x14. ◦ IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome value 0x18. ◦ IMPLEMENTATION DEFINED System registers, which are accessed using MRRS and MSRR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome value 0x14. • In AArch32 state, EL0 MCR and MRC access to the following encodings are trapped and reported using EC syndrome value 0x03: ◦ All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}. ◦ All coproc==p15, CRn==c10, opc1 =={0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}. ◦ All coproc==p15, CRn==c11, opc1=={0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

SP Interrupt Mask enable. When SCTLR_EL1.NMI is 1, controls whether PSTATE.SP acts as an interrupt mask, and controls the value of PSTATE.ALLINT on taking an exception to EL1.

SPINTMASK	Meaning
0b0	Does not cause PSTATE.SP to mask interrupts. PSTATE.ALLINT is set to 1 on taking an exception to EL1.
0b1	When PSTATE.SP is 1 and execution is at EL1, an IRQ or FIQ interrupt that is targeted to EL1 is masked regardless of any denotion of Superpriority. PSTATE.ALLINT is set to 0 on taking an exception to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]

When FEAT_NMI is implemented:

Non-maskable Interrupt enable.

NMI	Meaning
0b0	This control does not affect interrupt masking behavior.
0b1	This control enables all of the following: • The use of the PSTATE.ALLINT interrupt mask. • IRQ and FIQ interrupts to have Superpriority as an additional attribute. • PSTATE.SP to be used as an interrupt mask.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTP2, bit [60]

When FEAT_SME is implemented:

Traps instructions executed at EL0 that access TPIDR2_EL0 to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The exception is reported using EC syndrome value 0x18.

EnTP2	Meaning
0b0	This control causes execution of these instructions at EL0 to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO, bit [59]

When FEAT_MTE_STORE_ONLY is implemented:

Tag Checking Store Only.

TCSO	Meaning
0b0	This field has no effect on Tag checking.
0b1	Load instructions executed in EL1 are Tag Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO0, bit [58]

When FEAT_MTE_STORE_ONLY is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, Tag Checking Store Only in EL0.

TCSO0	Meaning
0b0	This field has no effect on Tag checking.
0b1	Load instructions executed in EL0 are Tag Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EPAN, bit [57]

When FEAT_PAN3 is implemented:

Enhanced Privileged Access Never. When PSTATE.PAN is 1, determines whether an EL1 data access to a page with stage 1 EL0 instruction access permission generates a Permission fault as a result of the Privileged Access Never mechanism.

EPAN	Meaning
0b0	No additional Permission faults are generated by this mechanism.
0b1	An EL1 data access to a page with stage 1 EL0 data access permission or stage 1 EL0 instruction access permission generates a Permission fault. Any speculative data accesses that would generate a Permission fault as a result of PSTATE.PAN = 1 if the accesses were not speculative, will not cause an allocation into a cache. When executing at EL1, this does not prevent unprivileged speculative accesses generated from the EL0 hardware-defined context from causing allocation into a cache.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnALS, bit [56]

When FEAT_LS64 is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps execution of an LD64B or ST64B instruction at EL0 to EL1.

EnALS	Meaning
0b0	Execution of an LD64B or ST64B instruction at EL0 is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

A trap of an LD64B or ST64B instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000002.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [55]
When FEAT_LS64_ACCDATA is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps execution of an ST64BV0 instruction at EL0 to EL1.

EnAS0	Meaning
0b0	Execution of an ST64BV0 instruction at EL0 is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV0 instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnASR, bit [54]
When FEAT_LS64_V is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps execution of an ST64BV instruction at EL0 to EL1.

EnASR	Meaning
0b0	Execution of an ST64BV instruction at EL0 is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME, bit [53]
When FEAT_TME is implemented:

Enables the Transactional Memory Extension at EL1.

TME	Meaning
0b0	Any attempt to execute a TSTART instruction at EL1 is trapped to EL1, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL1.
0b1	This control does not cause any TSTART instruction to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME0, bit [52]

When FEAT_TME is implemented:

Enables the Transactional Memory Extension at EL0.

TME0	Meaning
0b0	Any attempt to execute a TSTART instruction at EL0 is trapped to EL1, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL0.
0b1	This control does not cause any TSTART instruction to be trapped.

If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT, bit [51]

When FEAT_TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL1.

TMT	Meaning
0b0	This control does not cause any TSTART instruction to fail.
0b1	When the TSTART instruction is executed at EL1, the transaction fails with a TRIVIAL failure cause.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT0, bit [50]

When FEAT_TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL0.

TMT0	Meaning
0b0	This control does not cause any TSTART instruction to fail.
0b1	When the TSTART instruction is executed at EL0, the transaction fails with a TRIVIAL failure cause.

If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEL, bits [49:46]
When FEAT_TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when SCTLR_EL1.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by SCTLR_EL1.nTWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [45]
When FEAT_TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by SCTLR_EL1.nTWE.

TWEDEn	Meaning
0b0	The delay for taking the trap is IMPLEMENTATION DEFINED.
0b1	The delay for taking the trap is at least the number of cycles defined in SCTLR_EL1.TWEDEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSSBS, bit [44]
When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to EL1.
0b1	PSTATE.SSBS is set to 1 on an exception to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL1.

When SCR_EL3.ATA == 1 and HCR_EL2.ATA == 1, controls access to Allocation Tags and Tag Check operations in EL1.

ATA	Meaning
0b0	Access to Allocation Tags is prevented at EL1. Memory accesses at EL1 are not subject to a Tag Check operation.
0b1	This control does not prevent access to Allocation Tags at EL1. Tag Checked memory accesses at EL1 are subject to a Tag Check operation. The Tag Check operation depends on the type of tag at the memory being accessed: • For Allocation Tagged memory, an Allocation Tag Check operation. • If FEAT_MTE_CANONICAL_TAGS is implemented, for Canonically Tagged memory, a Canonical Tag Check operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA0, bit [42]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL0.

When SCR_EL3.ATA == 1, HCR_EL2.ATA == 1, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, controls access to Allocation Tags and Tag Check operations in EL0.

ATA0	Meaning
0b0	Access to Allocation Tags is prevented at EL0. Memory accesses at EL0 are not subject to a Tag Check operation.
0b1	This control does not prevent access to Allocation Tags at EL0. Tag Checked memory accesses at EL0 are subject to a Tag Check operation. The Tag Check operation depends on the type of tag at the memory being accessed: • For Allocation Tagged memory, an Allocation Tag Check operation. • If FEAT_MTE_CANONICAL_TAGS is implemented, for Canonically Tagged memory, a Canonical Tag Check operation.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF, bits [41:40]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL1. Controls the effect of Tag Check Faults due to Loads and Stores in EL1.

TCF	Meaning	Applies when
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	When FEAT_MTE3 is implemented

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF0, bits [39:38]
When FEAT_MTE2 is implemented:

Tag Check Fault in EL0. When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, controls the effect of Tag Check Faults due to Loads and Stores in EL0.

TCF0	Meaning	Applies when
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	When FEAT_MTE3 is implemented

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]
When FEAT_MTE_ASYNC is implemented:

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL1, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into TFSRE0_EL1 and TFSR_EL1 registers.

ITFSB	Meaning
0b0	Tag Check Faults are not synchronized on entry to EL1.
0b1	Tag Check Faults are synchronized on entry to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT1, bit [36]
When FEAT_BTI is implemented:

Configures the Branch Type compatibility of the implicit BTI behavior for the following instructions at EL1:

- PACIASP.
- PACIBSP.
- If FEAT_PAuth_LR is implemented, PACIASPPC.
- If FEAT_PAuth_LR is implemented, PACIBSPPC.

BT1	Meaning
0b0	When the PE is executing at EL1, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.jc.
0b1	When the PE is executing at EL1, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.c.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]
When FEAT_BTI is implemented:

Configures the Branch Type compatibility of the implicit BTI behavior for the following instructions at EL0:

- PACIASP.
- PACIBSP.
- If FEAT_PAuth_LR is implemented, PACIASPPC.
- If FEAT_PAuth_LR is implemented, PACIBSPPC.

BT0	Meaning
0b0	When the PE is executing at EL0, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.jc.
0b1	When the PE is executing at EL0, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.c.

When the value of the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the value of SCTLR_EL1.BT0 has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [34]
When FEAT_FPMR is implemented:

Enables direct and indirect accesses to FPMR from EL0.

When accesses to FPMR are disabled by this control:

- Direct accesses to FPMR from EL0 are trapped to EL1, or to EL2 when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1. These exceptions are reported using EC syndrome value 0x18.
- Execution of FP8 data-processing instructions that indirectly access FPMR is UNDEFINED at EL0.

EnFPM	Meaning
0b0	Direct and indirect accesses to FPMR are disabled at EL0.
0b1	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0, and the SCTLR_EL2.EnFPM control is used for this purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MSCEn, bit [33]

When FEAT_MOPS is implemented and !ELIsInHost(EL0):

Memory Copy and Memory Set instructions Enable. Enables execution of the Memory Copy and Memory Set instructions at EL0.

MSCEn	Meaning
0b0	Execution of the Memory Copy and Memory Set instructions is UNDEFINED at EL0.
0b1	This control does not cause any instructions to be UNDEFINED.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this bit is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [32]

When FEAT_CMOW is implemented:

Controls cache maintenance instruction permission for the following instructions executed at EL0.

- IC IVAU and DC CIVAC.
- If FEAT_MTE is implemented, DC CIGDVAC and DC CIGVAC.

CMOW	Meaning
0b0	These instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, do not generate a stage 1 permission fault.
0b1	If enabled as a result of SCTLR_EL1.UCI==1, these instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, generate a stage 1 permission fault.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

For this control, stage 1 has write permission if all of the following apply:

- AP[2] is 0 or DBM is 1 in the stage 1 descriptor.
- Where APTable is in use, APTable[1] is 0 for all levels of the translation table.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIA, bit [31]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIAKey_EL1 key, at EL1 and at EL0 in the EL1&0 translation regime.

EnIA	Meaning
0b0	Pointer authentication of instruction addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APIAKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APIAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. When pointer authentication is enabled, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIBKey_EL1 key, at EL1 and at EL0 in the EL1&0 translation regime.

EnIB	Meaning
0b0	Pointer authentication of instruction addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APIBKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APIBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. When pointer authentication is enabled, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]
When FEAT_LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

LSMAOE	Meaning
0b0	For all memory accesses at EL0, T32 and A32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1	The ordering and interrupt behavior of T32 and A32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0.

This bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

nTLSMD, bit [28]
When FEAT_LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

nTLSMD	Meaning
0b0	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.

This bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnDA, bit [27]
When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of data addresses, using the APDAKey_EL1 key, at EL1 and at EL0 in the EL1&0 translation regime.

EnDA	Meaning
0b0	Pointer authentication of data addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APDAKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APDAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. When pointer authentication is enabled, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]

Traps EL0 execution of cache maintenance instructions, to EL1, or to EL2 when it is implemented and enabled in the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using EC syndrome value 0x18, as follows:

- DC CVAU, DC CIVAC, DC CVAC, and IC IVAU.
- If FEAT_MTE is implemented, DC CIGVAC, DC CIGDVAC, DC CGVAC, and DC CGDVAC.
- If FEAT_DPB is implemented, DC CVAP.
- If FEAT_DPB and FEAT_MTE are implemented, DC CGVAP and DC CGDVAP.
- If FEAT_DPB2 is implemented, DC CVADP.
- If FEAT_DPB2 and FEAT_MTE are implemented, DC CGVADP and DC CGDVADP.
- If FEAT_OCCMO is implemented, DC CIVAOC, DC CIGDVAOC, DC CVAOC and DC CGDVAOC.

UCI	Meaning
0b0	For each of the specified instructions, if the execution of the instruction can be trapped, access at EL0 using AArch64 is trapped.
0b1	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EE, bit [25]

When FEAT_MixedEnd is implemented:

Endianness of data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are little-endian.
0b1	Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are big-endian.

The EE bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Endianness of data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are little-endian.
0b1	Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are big-endian.

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

E0E, bit [24]

When FEAT_MixedEndEL0 is implemented:

Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

If an implementation only supports Little-endian accesses at EL0, then this bit is RES0. This option is not permitted when SCTLR_EL1.EE is RES1 .

If an implementation only supports Big-endian accesses at EL0, then this bit is RES1. This option is not permitted when SCTLR_EL1.EE is RES0.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

SPAN, bit [23]

When FEAT_PAN is implemented:

Set Privileged Access Never, on taking an exception to EL1.

SPAN	Meaning
0b0	PSTATE.PAN is set to 1 on taking an exception to EL1.
0b1	The value of PSTATE.PAN is left unchanged on taking an exception to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EIS, bit [22]

When FEAT_ExS is implemented:

Exception Entry is Context Synchronizing.

EIS	Meaning
0b0	The taking of an exception to EL1 is not a context synchronizing event.
0b1	The taking of an exception to EL1 is a context synchronizing event.

If SCTLR_EL1.EIS is set to 0b0:

- Indirect writes to ESR_EL1, FAR_EL1, SPSR_EL1, ELR_EL1 are synchronized on exception entry to EL1, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.
- Some exception entries reported are not considered IFBEs per the memory model.

The following are not affected by the value of SCTLR_EL1.EIS:

- Changes to the PSTATE information on entry to EL1.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- The memory model requirement for exception entry to generate an Instruction Fetch Barrier Effect for some exception entries. See Basic definitions for the list of exception entries.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]

When FEAT_IESB is implemented:

Implicit Error Synchronization event enable. Possible values are:

IESB	Meaning
0b0	Disabled.
0b1	An implicit error synchronization event is added: • At each exception taken to EL1. • Before the operational pseudocode of each ERET instruction executed at EL1.

If FEAT_DoubleFault2 is implemented, the PE is in Non-debug state, and the Effective value of SCTLR2_EL1.NMEA is 1, then SCTLR_EL1.IESB is ignored and the PE behaves as if SCTLR_EL1.IESB is 1 for all purposes other than direct read of the register.

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each `DCPSX` instruction taken to EL1 and before each `DRPS` instruction executed at EL1, in addition to the other cases where it is added.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSCXT, bit [20]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap EL0 Access to the SCXTNUM_EL0 register, when EL0 is using AArch64.

TSCXT	Meaning
`0b0`	EL0 access to SCXTNUM_EL0 is not disabled by this mechanism.
`0b1`	EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The value of SCXTNUM_EL0 is treated as 0.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL1&0 translation regime, this bit can restrict execute permissions on writeable pages.

WXN	Meaning
`0b0`	This control has no effect on memory access permissions.
`0b1`	In the EL1&0 translation regime, any region of memory that is writeable at EL1 is XN at EL1, and any region of memory that is writeable at EL0 is XN at EL0.

This bit applies only when SCTLR_EL1.M bit is set.

The WXN bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

This field is RES0 if TCR2_EL1.PIE is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

Traps EL0 execution of WFE instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states, reported using EC syndrome value `0x01`.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

nTWE	Meaning
0b0	Any attempt to execute a WFE instruction at EL0 is trapped, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Traps EL0 execution of WFI instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states, reported using EC syndrome value 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

nTWI	Meaning
0b0	Any attempt to execute a WFI instruction at EL0 is trapped, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UCT, bit [15]

Traps EL0 accesses to the CTR_EL0 to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using EC syndrome value 0x18.

UCT	Meaning
0b0	Accesses to the CTR_EL0 from EL0 using AArch64 are trapped.
0b1	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZE, bit [14]

Traps EL0 execution of DC ZVA instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using EC syndrome value 0x18.

If FEAT_MTE is implemented, this trap also applies to DC GVA and DC GZVA.

DZE	Meaning
0b0	Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped. Reading DCZID_EL0.DZP from EL0 returns 1, indicating that the instructions this trap applies to are not supported.
0b1	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of data addresses, using the APDBKey_EL1 key, at EL1 and at EL0 in the EL1&0 translation regime.

EnDB	Meaning
0b0	Pointer authentication of data addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APDBKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APDBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. When pointer authentication is enabled, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Stage 1 instruction access Cacheability control, for accesses at EL0 and EL1:

I	Meaning
0b0	All instruction access to Stage 1 Normal memory from EL0 and EL1 are Stage 1 Non-cacheable. If the value of SCTLR_EL1.M is 0, instruction accesses from stage 1 of the EL1&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	This control has no effect on the Stage 1 Cacheability of instruction access to Stage 1 Normal memory from EL0 and EL1. If the value of SCTLR_EL1.M is 0, instruction accesses from stage 1 of the EL1&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

When the value of the HCR_EL2.DC bit is 1, then instruction access to Normal memory from EL0 and EL1 are Cacheable regardless of the value of the SCTLR_EL1.I bit.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EOS, bit [11]

When FEAT_ExS is implemented:

Exception Exit is Context Synchronizing.

EOS	Meaning
0b0	An exception return from EL1 is not a context synchronizing event
0b1	An exception return from EL1 is a context synchronizing event

If SCTLR_EL1.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL1.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL1 and ELR_EL1 on exception return is synchronized.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnRCTX, bit [10]

When FEAT_SPECRES is implemented:

Enable EL0 access to the following System instructions:

- CFPRCTX, DVPRCTX and CPPRCTX instructions.
- If FEAT_SPECRES2 is implemented, COSPRCTX.
- CFP RCTX, DVP RCTX and CPP RCTX instructions.
- If FEAT_SPECRES2 is implemented, COSP RCTX.

EnRCTX	Meaning
0b0	EL0 access to these instructions is disabled, and these instructions are trapped to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1.
0b1	EL0 access to these instructions is enabled.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UMA, bit [9]

User Mask Access. Traps EL0 execution of MSR and MRS instructions that access the PSTATE.{D, A, I, F} masks to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using EC syndrome value 0x18.

UMA	Meaning
0b0	Any attempt at EL0 using AArch64 to execute an MRS, MSR(REGISTER), or MSR(IMMEDIATE) instruction that accesses the DAIF is trapped.
0b1	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the value of this bit is treated as 0 for all purposes other than reading the value of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SED, bit [8]

When FEAT_AA32EL0 is implemented:

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

SED	Meaning
0b0	SETEND instruction execution is enabled at EL0 using AArch32.
0b1	SETEND instructions are UNDEFINED at EL0 using AArch32 and any attempt at EL0 to access a SETEND instruction generates an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, reported using EC syndrome value 0x00.

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

ITD, bit [7]

When FEAT_AA32EL0 is implemented:

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.

ITD	Meaning
0b0	All IT instruction functionality is enabled at EL0 using AArch32.
0b1	Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED and generates an exception, reported using EC syndrome value 0x00, to EL1 or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1: • All encodings of the IT instruction with hw1[3:0]!=1000. • All encodings of the subsequent instruction with the following values for hw1: ◦ 0b11xxxxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. ◦ 0b1011xxxxxxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions'. ◦ 0b10100xxxxxxxxxx: ADD Rd, PC, #imm ◦ 0b01001xxxxxxxxxx: LDR Rd, [PC, #imm] ◦ 0b0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. ◦ 0b010001xx1xxxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn. These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block. It is IMPLEMENTATION DEFINED whether the IT instruction is treated as: • A 16-bit instruction, that can only be followed by another 16-bit instruction. • The first half of a 32-bit instruction. This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED. An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information, see 'Changes to an ITD control by an instruction in an IT block'.

ITD is optional, but if it is implemented in the SCTLR_EL1 then it must also be implemented in the SCTLR_EL2, HSCTLR, and SCTLR.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement ITD, access to this field is **RAZ/WI**.

Otherwise:

Reserved, RES1.

nAA, bit [6]

When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL1 and EL0 under certain conditions.

The following instructions generate an Alignment fault if all bytes being accessed are not within a single naturally aligned 16-byte quantity, for access:

- LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH.
- STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH.
- If FEAT_LRCPC3 is implemented, the post index versions of LDAPR and the pre index versions of STLR.
- If FEAT_LRCPC3 and Advanced SIMD and floating-point instructions are implemented, LDAPUR (SIMD&FP), LDAP1 (SIMD&FP), STLUR (SIMD&FP), and STL1 (SIMD&FP).

If FEAT_LRCPC3 is implemented, the following instructions generate an Alignment fault if all bytes being accessed for a single register are not within a single naturally aligned 16-byte quantity, for access:

- LDIAPP, STILP.

nAA	Meaning
0b0	Unaligned accesses by the specified instructions generate an Alignment fault.
0b1	This control does not generate Alignment faults.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CP15BEN, bit [5]

When FEAT_AA32EL0 is implemented:

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL0:

CP15BEN	Meaning
0b0	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED and generates an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The exception is reported using EC syndrome value 0x00.
0b1	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

CP15BEN is optional, but if it is implemented in the SCTLR_EL1 then it must also be implemented in the SCTLR_EL2, HSCTLR, and SCTLR.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement CP15BEN, access to this field is **RAO/WI**.

Otherwise:

Reserved, RES0.

SA0, bit [4]

SP Alignment check enable for EL0. When set to 1, if a load or store instruction executed at EL0 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL1 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Stage 1 Cacheability control, for data accesses.

C	Meaning
0b0	All data access to Stage 1 Normal memory from EL0 and EL1, and all Normal memory accesses from unified cache to the EL1&0 Stage 1 translation tables, are treated as Stage 1 Non-cacheable.
0b1	This control has no effect on the Stage 1 Cacheability of: • Data access to Normal memory from EL0 and EL1. • Normal memory accesses to the EL1&0 Stage 1 translation tables.

When the Effective value of the HCR_EL2.DC bit in the current Security state is 1, the PE ignores SCTLR_EL1.C. This means that EL0 and EL1 data accesses to Normal memory are Cacheable.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL1 and EL0.

A	Meaning
0b0	Alignment fault checking is disabled when executing at EL1 or EL0. Alignment checks on some instructions are not disabled by this control. For more information, see 'Alignment of data accesses'.
0b1	Alignment fault checking is enabled when executing at EL1 or EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the value of this bit is treated as 0 for all purposes other than reading the value of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL1&0 stage 1 address translation.

M	Meaning
0b0	EL1&0 stage 1 address translation disabled. See the SCTLR_EL1.I field for the behavior of instruction accesses to Normal memory.
0b1	EL1&0 stage 1 address translation enabled.

If the Effective value of HCR_EL2.{DC, TGE} in the current Security state is not {0, 0} then the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of the field.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset:
    - When the highest implemented Exception level is EL1, this field resets to `'0'`.
    - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing SCTLR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name SCTLR_EL1 or SCTLR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to SCTLR_EL1 are masked by SCTLRMASK_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x110];
    else
        X[t, 64] = SCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLR_EL2;
    else
        X[t, 64] = SCTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR_EL1;
```

```
MSR SCTLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x110] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL1 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL1()) OR (SCTLR_EL1 AND
EffectiveSCTLRMASK_EL1());
        else
            SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL2 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL2()) OR (SCTLR_EL2 AND
EffectiveSCTLRMASK_EL2());
        else
            SCTLR_EL2 = X[t, 64];
    else
        SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, SCTLR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x110];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR SCTLR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2

0b11	0b101	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x110] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        SCTLR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SCTLR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_SRMASK is implemented

```
MRS <Xt>, SCTLRALIAS_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nSCTLRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x110];
    else
        X[t, 64] = SCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLR_EL2;
    else
        X[t, 64] = SCTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR_EL1;
```

When FEAT_SRMASK is implemented

```
MSR SCTLRALIAS_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nSCTLRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x110] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL1 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL1()) OR (SCTLR_EL1 AND
EffectiveSCTLRMASK_EL1());
        else
            SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL2 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL2()) OR (SCTLR_EL2 AND
EffectiveSCTLRMASK_EL2());
        else
            SCTLR_EL2 = X[t, 64];
    else
        SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR_EL2, System Control Register (EL2)

The SCTLR_EL2 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL2.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, these controls apply also to execution at EL0.

Configuration

AArch64 System register SCTLR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HSCTLR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR_EL2 are UNDEFINED.

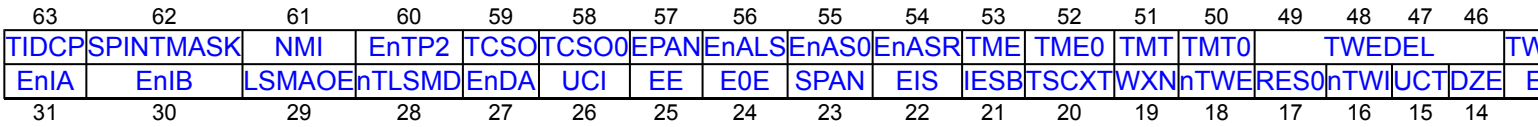
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

SCTLR_EL2 is a 64-bit register.

Field descriptions



TIDCP, bit [63]

When FEAT_TIDCP1 is implemented and ELIsInHost(EL2):

Trap IMPLEMENTATION DEFINED functionality. Traps EL0 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL2.

TIDCP	Meaning
0b0	No instructions accessing the System register or System instruction spaces are trapped by this mechanism.
0b1	If HCR_EL2.TGE==0, no instructions accessing the System register or System instruction spaces are trapped by this mechanism. If HCR_EL2.TGE==1, instructions accessing the following System register or System instruction spaces are trapped to EL2 by this mechanism: • In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped: ◦ IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}, and are reported using EC syndrome value 0x18. ◦ IMPLEMENTATION DEFINED System instructions, which are accessed using SYSP, with CRn == {11, 15}, and are reported using EC syndrome value 0x14. ◦ IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome value 0x18. ◦ IMPLEMENTATION DEFINED System registers, which are accessed using MRRS and MSRR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome value 0x14. • In AArch32 state, EL0 MCR and MRC accesses to the following encodings are trapped and reported using EC syndrome value 0x03: ◦ All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}. ◦ All coproc==p15, CRn==c10, opc1 =={0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}. ◦ All coproc==p15, CRn==c11, opc1=={0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

SP Interrupt Mask enable. When SCTLR_EL2.NMI is 1, controls whether PSTATE.SP acts as an interrupt mask, and controls the value of PSTATE.ALLINT on taking an exception to EL2.

SPINTMASK	Meaning
0b0	Does not cause PSTATE.SP to mask interrupts. PSTATE.ALLINT is set to 1 on taking an exception to EL2.
0b1	When PSTATE.SP is 1 and execution is at EL2, an IRQ or FIQ interrupt that is targeted to EL2 is masked regardless of any denotation of Superpriority. PSTATE.ALLINT is set to 0 on taking an exception to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]

When FEAT_NMI is implemented:

Non-maskable Interrupt enable.

NMI	Meaning
0b0	This control does not affect interrupt masking behavior.
0b1	This control enables all of the following: • The use of the PSTATE.ALLINT interrupt mask. • IRQ and FIQ interrupts to have Superpriority as an additional attribute. • PSTATE.SP to be used as an interrupt mask.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTP2, bit [60]

When FEAT_SME is implemented and ELIsInHost(EL2):

Traps instructions executed at EL0 that access TPIDR2_EL0 to EL2 when EL2 is implemented and enabled for the current Security state. The exception is reported using EC syndrome value 0x18.

EnTP2	Meaning
0b0	This control causes execution of these instructions at EL0 to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO, bit [59]

When FEAT_MTE_STORE_ONLY is implemented:

Tag Checking Store Only.

TCSO	Meaning
0b0	This field has no effect on Tag checking.
0b1	Load instructions executed in EL2 are Tag Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO0, bit [58]

When FEAT_MTE_STORE_ONLY is implemented and ELIsInHost(EL2):

Tag Checking Store Only in EL0.

TCSO0	Meaning
0b0	This field has no effect on Tag checking.
0b1	Load instructions executed in EL0 are Tag Unchecked.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EPAN, bit [57]

When FEAT_PAN3 is implemented and ELIsInHost(EL2):

Enhanced Privileged Access Never. When PSTATE.PAN is 1, determines whether an EL2 data access to a page with EL0 instruction access permission generates a Permission fault as a result of the Privileged Access Never mechanism.

EPAN	Meaning
0b0	No additional Permission faults are generated by this mechanism.
0b1	An EL2 data access to a page with stage 1 EL0 data access permission or stage 1 EL0 instruction access permission generates a Permission fault. Any speculative data accesses that would generate a Permission fault as a result of PSTATE.PAN = 1 if the accesses were not speculative, will not cause an allocation into a cache. When executing at EL2, and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this does not prevent unprivileged speculative accesses generated from the EL0 hardware-defined context from causing allocation into a cache.

Note

The value of HCR_EL2.TGE does not change the effect of this field on privileged accesses.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnALS, bit [56]

When FEAT_LS64 is implemented and ELIsInHost(EL2):

Traps execution of an LD64B or ST64B instruction at EL0 to EL2.

EnALS	Meaning
0b0	Execution of an LD64B or ST64B instruction at EL0 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

A trap of an LD64B or ST64B instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000002.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [55]
When FEAT_LS64_ACCDATA is implemented and ELIsInHost(EL2):

Traps execution of an ST64BV0 instruction at EL0 to EL2.

EnAS0	Meaning
0b0	Execution of an ST64BV0 instruction at EL0 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

A trap of an ST64BV0 instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnASR, bit [54]
When FEAT_LS64_V is implemented and ELIsInHost(EL2):

Traps execution of an ST64BV instruction at EL0 to EL2.

EnASR	Meaning
0b0	Execution of an ST64BV instruction at EL0 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

A trap of an ST64BV instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME, bit [53]
When FEAT_TME is implemented:

Enables the Transactional Memory Extension at EL2.

TME	Meaning
0b0	Any attempt to execute a TSTART instruction at EL2 is trapped, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL2.
0b1	This control does not cause any TSTART instruction to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME0, bit [52]

When FEAT_TME is implemented and ELIsInHost(EL2):

Enables the Transactional Memory Extension at EL0.

TME0	Meaning
0b0	Any attempt to execute a TSTART instruction at EL0 is trapped to EL2, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL0.
0b1	This control does not cause any TSTART instruction to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT, bit [51]

When FEAT_TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL2.

TMT	Meaning
0b0	This control does not cause any TSTART instruction to fail.
0b1	When the TSTART instruction is executed at EL2, the transaction fails with a TRIVIAL failure cause.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT0, bit [50]

When FEAT_TME is implemented and ELIsInHost(EL2):

Forces a trivial implementation of the Transactional Memory Extension at EL0.

TMT0	Meaning
0b0	This control does not cause any TSTART instruction to fail.
0b1	When the TSTART instruction is executed at EL0, the transaction fails with a TRIVIAL failure cause.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEL, bits [49:46]
When FEAT_TWED is implemented and ELIsInHost(EL2):

TWE Delay. A 4-bit unsigned number that, when SCTLR_EL2.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE caused by SCTLR_EL2.nTWE as $2^{(TWEDEL + 8)}$ cycles.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [45]
When FEAT_TWED is implemented and ELIsInHost(EL2):

TWE Delay Enable. Enables a configurable delayed trap of the WFE instruction caused by SCTLR_EL2.nTWE.

TWEDEn	Meaning
0b0	The delay for taking a WFE trap is IMPLEMENTATION DEFINED.
0b1	The delay for taking a WFE trap is at least the number of cycles defined in SCTLR_EL2.TWEDEL.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSSBS, bit [44]
When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on exception entry.

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to EL2.
0b1	PSTATE.SSBS is set to 1 on an exception to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]
When FEAT_MTE2 is implemented:

Allocation Tag Access in EL2.

When SCR_EL3.ATA is 1, controls access to Allocation Tags and Tag Check operations in EL2.

ATA	Meaning
0b0	Access to Allocation Tags is prevented at EL2. Memory accesses at EL2 are not subject to a Tag Check operation.
0b1	This control does not prevent access to Allocation Tags at EL2. Tag Checked memory accesses at EL2 are subject to a Tag Check operation. The Tag Check operation depends on the type of tag at the memory being accessed: • For Allocation Tagged memory, an Allocation Tag Check operation. • If FEAT_MTE_CANONICAL_TAGS is implemented, for Canonically Tagged memory, a Canonical Tag Check operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA0, bit [42]
When FEAT_MTE2 is implemented and ELIsInHost(EL2):

Allocation Tag Access in EL0.

When SCR_EL3.ATA is 1, controls access to Allocation Tags and Tag Check operations in EL0.

ATA0	Meaning
0b0	Access to Allocation Tags is prevented at EL0. Memory accesses at EL0 are not subject to a Tag Check operation.
0b1	This control does not prevent access to Allocation Tags at EL0. Tag Checked memory accesses at EL0 are subject to a Tag Check operation. The Tag Check operation depends on the type of tag at the memory being accessed: • For Allocation Tagged memory, an Allocation Tag Check operation. • If FEAT_MTE_CANONICAL_TAGS is implemented, for Canonically Tagged memory, a Canonical Tag Check operation.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF, bits [41:40]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL2. Controls the effect of Tag Check Faults due to Loads and Stores in EL2.

TCF	Meaning	Applies when
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	When FEAT_MTE3 is implemented

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF0, bits [39:38]

When FEAT_MTE2 is implemented and ELIsInHost(EL2):

Tag Check Fault in EL0. Controls the effect of Tag Check Faults due to Loads and Stores in EL0.

TCF0	Meaning	Applies when
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	When FEAT_MTE3 is implemented

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]

When FEAT_MTE_ASYNC is implemented:

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL2, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into TFSRE0_EL1, TFSR_EL1, and TFSR_EL2 registers.

ITFSB	Meaning
0b0	Tag Check Faults are not synchronized on entry to EL2.
0b1	Tag Check Faults are synchronized on entry to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bit [36]

When FEAT_BTI is implemented:

Indicates the Branch Type compatibility of the implicit BTI behavior for the following instructions at EL2:

- PACIASP.
- PACIBSP.
- If FEAT_PAuth_LR is implemented, PACIASPPC.
- If FEAT_PAuth_LR is implemented, PACIBSPPC.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit is named BT1.

BT	Meaning
0b0	When the PE is executing at EL2, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.jc.
0b1	When the PE is executing at EL2, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.c.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]

When FEAT_BTI is implemented and ELIsInHost(EL2):

Indicates the Branch Type compatibility of the implicit BTI behavior for the following instructions at EL0:

- PACIASP.
- PACIBSP.
- If FEAT_PAuth_LR is implemented, PACIASPPC.
- If FEAT_PAuth_LR is implemented, PACIBSPPC.

BT0	Meaning
0b0	When the PE is executing at EL0, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.jc.
0b1	When the PE is executing at EL0, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.c.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [34]
When FEAT_FPMR is implemented and ELIsInHost(EL0):

Enables direct and indirect accesses to FPMR from EL0.

When accesses to FPMR are disabled by this control:

- Direct accesses to FPMR from EL0 are trapped to EL2 and reported using EC syndrome value `0x18`.
- Execution of FP8 data-processing instructions that indirectly access FPMR is UNDEFINED at EL0.

EnFPM	Meaning
`0b0`	Direct and indirect accesses to FPMR are disabled at EL0.
`0b1`	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

If EL2 is not implemented or is disabled in the current Security state, the Effective value of this field is `0b1`.

When HCR_EL2.TGE is 0, this field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MSCEn, bit [33]
When FEAT_MOPS is implemented and ELIsInHost(EL2):

Memory Copy and Memory Set instructions Enable. Enables execution of the Memory Copy and Memory Set instructions at EL0.

MSCEn	Meaning
`0b0`	Execution of the Memory Copy and Memory Set instructions is UNDEFINED at EL0.
`0b1`	This control does not cause any instructions to be UNDEFINED.

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

When FEAT_MOPS is implemented and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the Effective value of this bit is `0b1`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [32]

When FEAT_CMOW is implemented and ELIsInHost(EL2):

Controls cache maintenance instruction permission for the following instructions executed at EL0.

- IC IVAU and DC CIVAC.
- If FEAT_MTE is implemented, DC CIGDVAC and DC CIGVAC.

CMOW	Meaning
0b0	These instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, do not generate a stage 1 permission fault.
0b1	If enabled as a result of SCTLR_EL2.UCI==1, these instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, generate a stage 1 permission fault.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

For this control, stage 1 has write permission if all of the following apply:

- AP[2] is 0 or DBM is 1 in the stage 1 descriptor.
- Where APTable is in use, APTable[1] is 0 for all levels of the translation table.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIA, bit [31]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIAKey_EL1 key, at EL2 and at EL0 in the EL2&0 translation regime.

EnIA	Meaning
0b0	Pointer authentication of instruction addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APIAKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APIAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. When pointer authentication is enabled, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIBKey_EL1 key, at EL2 and at EL0 in the EL2&0 translation regime.

EnIB	Meaning
0b0	Pointer authentication of instruction addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APIBKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APIBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. When pointer authentication is enabled, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]

When FEAT_LSMAOC is implemented and ELIsInHost(EL2):

Load Multiple and Store Multiple Atomicity and Ordering Enable.

LSMAOE	Meaning
0b0	For all memory accesses at EL0, T32 and A32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1	The ordering and interrupt behavior of T32 and A32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

nTLSMD, bit [28]

When FEAT_LSMAOC is implemented and ELIsInHost(EL2):

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

nTLSMD	Meaning
0b0	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnDA, bit [27]
When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of data addresses, using the APDAKey_EL1 key, at EL2 and at EL0 in the EL2&0 translation regime.

EnDA	Meaning
0b0	Pointer authentication of data addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APDAKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APDAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. When pointer authentication is enabled, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]
When ELIsInHost(EL2):

Traps execution of cache maintenance instructions at EL0 to EL2, from AArch64 state only, reported using EC syndrome value 0x18, as follows:

- DC CVAU, DC CIVAC, DC CVAC, and IC IVAU.
- If FEAT_MTE is implemented, DC CIGVAC, DC CIGDVAC, DC CGVAC, and DC CGDVAC.
- If FEAT_DPB is implemented, DC CVAP.
- If FEAT_DPB and FEAT_MTE are implemented, DC CGVAP and DC CGDVAP.
- If FEAT_DPB2 is implemented, DC CVADP.
- If FEAT_DPB2 and FEAT_MTE are implemented, DC CGVADP and DC CGDVADP.
- If FEAT_OCCMO is implemented, DC CIVAOC, DC CIGDVAOC, DC CVAOC and DC CGDVAOC.

UCI	Meaning
0b0	For each of the specified instructions, if the execution of the instruction can be trapped, access at EL0 using AArch64 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bit [25]
When FEAT_MixedEnd is implemented:

Endianness of data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime are little-endian.
0b1	Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime are big-endian.

The EE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Endianness of data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime are are little-endian.
0b1	Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime are are big-endian.

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

E0E, bit [24]
When ELIsInHost(EL2) and FEAT_MixedEndEL0 is implemented:

Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

If an implementation only supports Little-endian accesses at EL0, then this bit is RES0. This option is not permitted when SCTLR_EL1.EE is RES1.

If an implementation only supports Big-endian accesses at EL0, then this bit is RES1. This option is not permitted when SCTLR_EL1.EE is RES0.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SPAN, bit [23]

When ELIsInHost(EL2):

Set Privileged Access Never, on taking an exception to EL2.

SPAN	Meaning
0b0	PSTATE.PAN is set to 1 on taking an exception to EL2.
0b1	The value of PSTATE.PAN is left unchanged on taking an exception to EL2.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EIS, bit [22]

When FEAT_ExS is implemented:

Exception entry is a context synchronization event.

EIS	Meaning
0b0	The taking of an exception to EL2 is not a context synchronization event.
0b1	The taking of an exception to EL2 is a context synchronization event.

If SCTLR_EL2.EIS is set to 0b0:

- Indirect writes to ESR_EL2, FAR_EL2, SPSR_EL2, ELR_EL2, and HPFAR_EL2 are synchronized on exception entry to EL2, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.
- Some exception entries reported are not considered IFBEs per the memory model.

The following are not affected by the value of SCTLR_EL2.EIS:

- Changes to the PSTATE information on entry to EL2.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores, and data processing instructions.
- The memory model requirement for exception entry to generate an Instruction Fetch Barrier Effect for some exception entries. See Basic definitions for the list of exception entries.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]

When FEAT_IESB is implemented:

Implicit Error Synchronization event enable.

IESB	Meaning
0b0	Disabled.
0b1	An implicit error synchronization event is added: • At each exception taken to EL2. • Before the operational pseudocode of each ERET instruction executed at EL2.

If FEAT_DoubleFault2 is implemented, the PE is in Non-debug state, and the Effective value of SCTLR2_EL2.NMEA is 1, then SCTLR_EL2.IESB is ignored and the PE behaves as if SCTLR_EL2.IESB is 1 for all purposes other than direct read of the register.

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSX instruction taken to EL2 and before each DRPS instruction executed at EL2, in addition to the other cases where it is added.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSCXT, bit [20]

When (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented) and ELIsInHost(EL2):

Trap EL0 Access to the SCXTNUM_EL0 register, when EL0 is using AArch64.

TSCXT	Meaning
0b0	EL0 access to SCXTNUM_EL0 is not disabled by this mechanism.
0b1	EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL2, and the SCXTNUM_EL0 value is treated as 0.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_CSV2_2 is not implemented, FEAT_CSV2_1p2 is not implemented, and ELIsInHost(EL0):

Reserved, RES1.

Otherwise:

Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL2 or EL2&0 translation regime, this bit can restrict execute permissions on writeable pages.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	In the EL2 or EL2&0 translation regime, any region of memory that is writeable at EL2 is XN at EL2. In the EL2&0 translation regime, any region of memory that is writeable at EL0 is XN at EL0.

This bit applies only when SCTLR_EL2.M bit is set.

The WXN bit is permitted to be cached in a TLB.

This field is RES0 if TCR2_EL2.PIE is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

When ELIsInHost(EL2):

Traps execution of WFE instructions at EL0 to EL2, from both Execution states.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

nTWE	Meaning
0b0	Any attempt to execute a WFE instruction at EL0 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

When ELIsInHost(EL2):

Traps execution of WFI instructions at EL0 to EL2, from both Execution states.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

nTWI	Meaning
0b0	Any attempt to execute a WFI instruction at EL0 is trapped EL2, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

UCT, bit [15]

When ELIsInHost(EL2):

Traps EL0 accesses to the CTR_EL0 to EL2, from AArch64 state only.

UCT	Meaning
0b0	Accesses to the CTR_EL0 from EL0 using AArch64 are trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DZE, bit [14]

When ELIsInHost(EL2):

Traps execution of DC ZVA instructions at EL0 to EL2, from AArch64 state only.

If FEAT_MTE is implemented, this trap also applies to DC GVA and DC GZVA.

DZE	Meaning
0b0	Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped to EL2. Reading DCZID_EL0.DZP from EL0 returns 1, indicating that the instructions that this trap applies to are not supported.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of data addresses, using the APDBKey_EL1 key, at EL2 and at EL0 in the EL2&0 translation regime.

EnDB	Meaning
0b0	Pointer authentication of data addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APDBKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APDBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. When pointer authentication is enabled, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL2 and, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL0.

I	Meaning
0b0	All instruction accesses to Normal memory from EL2 are Non-cacheable for all levels of instruction and unified cache. When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, all instruction accesses to Normal memory from EL0 are Non-cacheable for all levels of instruction and unified cache. If SCTLR_EL2.M is 0, instruction accesses from stage 1 of the EL2 or EL2&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	This control has no effect on the Cacheability of instruction access to Normal memory from EL2 and, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, instruction access to Normal memory from EL0. If the value of SCTLR_EL2.M is 0, instruction accesses from stage 1 of the EL2 or EL2&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

This bit has no effect on the EL3 translation regime.

When EL2 is disabled in the current Security state or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, this bit has no effect on the EL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EOS, bit [11]
When FEAT_ExS is implemented:

Exception exit is a context synchronization event.

EOS	Meaning
0b0	An exception return from EL2 is not a context synchronization event.
0b1	An exception return from EL2 is a context synchronization event.

If SCTLR_EL2.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL2.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL2 and ELR_EL2 on exception return is synchronized.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores, and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnRCTX, bit [10]
When FEAT_SPECRES is implemented and ELIsInHost(EL2):

Enable EL0 access to the following System instructions:

- CFPRCTX, DVPRCTX and CPPRCTX instructions.

- If FEAT_SPECRES2 is implemented, COSPRCTX.
- CFP RCTX, DVP RCTX and CPP RCTX instructions.
- If FEAT_SPECRES2 is implemented, COSP RCTX.

EnRCTX	Meaning
0b0	EL0 access to these instructions is disabled, and these instructions are trapped to EL2.
0b1	EL0 access to these instructions is enabled.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [9]

Reserved, RES0.

SED, bit [8]

When FEAT_AA32EL0 is implemented and ELIsInHost(EL2):

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

SED	Meaning
0b0	SETEND instruction execution is enabled at EL0 using AArch32.
0b1	SETEND instructions are UNDEFINED at EL0 using AArch32.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_AA32EL0 is not implemented and ELIsInHost(EL2):

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

Access to this field is **RES1**.

Otherwise:

Reserved, RES0.

ITD, bit [7]

When FEAT_AA32EL0 is implemented and ELIsInHost(EL2):

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.

ITD	Meaning
0b0	All IT instruction functionality is enabled at EL0 using AArch32.
0b1	Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED : • All encodings of the IT instruction with hw1[3:0]!=1000. • All encodings of the subsequent instruction with the following values for hw1: ◦ 0b11xxxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. ◦ 0b1011xxxxxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions'. ◦ 0b10100xxxxxxxxxx: ADD Rd, PC, #imm ◦ 0b01001xxxxxxxxxx: LDR Rd, [PC, #imm] ◦ 0b0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. ◦ 0b010001xx1xxxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers UNPREDICTABLE cases with BLX Rn. These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block. It is IMPLEMENTATION DEFINED whether the IT instruction is treated as: • A 16-bit instruction, that can only be followed by another 16-bit instruction. • The first half of a 32-bit instruction. This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED. An implementation might vary dynamically whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information see 'Changes to an ITD control by an instruction in an IT block'.

ITD is optional, but if it is implemented in the SCTLR_EL2 then it must also be implemented in the SCTLR_EL1, HSCTLR, and SCTLR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement ITD, access to this field is **RAZ/WI**.

When FEAT_AA32EL0 is not implemented and ELIsInHost(EL2):

Reserved, RES1.

Otherwise:

Reserved, RES0.

nAA, bit [6]
When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults under certain conditions at EL2, and, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL0.

The following instructions generate an Alignment fault if all bytes being accessed are not within a single naturally aligned 16-byte quantity, for access:

- LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH.
- STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH.

- If FEAT_LRCPC3 is implemented, the post index versions of LDAPR and the pre index versions of STLR.
- If FEAT_LRCPC3 and Advanced SIMD and floating-point instructions are implemented, LDAPUR (SIMD&FP), LDAP1 (SIMD&FP), STLUR (SIMD&FP), and STL1 (SIMD&FP).

If FEAT_LRCPC3 is implemented, the following instructions generate an Alignment fault if all bytes being accessed for a single register are not within a single naturally aligned 16-byte quantity, for access:

- LDIAPP, STILP.

nAA	Meaning
0b0	Unaligned accesses by the specified instructions generate an Alignment fault.
0b1	Unaligned accesses by the specified instructions do not generate an Alignment fault.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CP15BEN, bit [5]
When FEAT_AA32EL0 is implemented and ELIsInHost(EL2):

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL0:

CP15BEN	Meaning
0b0	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED.
0b1	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

CP15BEN is optional, but if it is implemented in the SCTLR_EL2 then it must also be implemented in the SCTLR_EL1, HSCTLR, and SCTLR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_AA32EL0 is not implemented and ELIsInHost(EL2):

Access to this field is **RES0**.

Otherwise:

Reserved, RES1.

SA0, bit [4]
When ELIsInHost(EL2):

SP Alignment check enable for EL0. When set to 1, if a load or store instruction executed at EL0 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL2 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Data access Cacheability control, for accesses at EL2 and, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL0

C	Meaning
0b0	The following are Non-cacheable for all levels of data and unified cache: • Data accesses to Normal memory from EL2. • When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, Normal memory accesses to the EL2 translation tables. • When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}: ◦ Data accesses to Normal memory from EL0. ◦ Normal memory accesses to the EL2&0 translation tables.
0b1	This control has no effect on the Cacheability of: • Data access to Normal memory from EL2. • When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, Normal memory accesses to the EL2 translation tables. • When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}: ◦ Data accesses to Normal memory from EL0. ◦ Normal memory accesses to the EL2&0 translation tables.

This bit has no effect on the EL3 translation regime.

When EL2 is disabled in the current Security state or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, this bit has no effect on the EL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL2 and, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL0.

A	Meaning
0b0	Alignment fault checking is disabled when executing at EL2. When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, alignment fault checking disabled when executing at EL0. Alignment checks on some instructions are not disabled by this control. For more information, see 'Alignment of data accesses'.
0b1	Alignment fault checking is enabled when executing at EL2. When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, alignment fault checking enabled when executing at EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL2 or EL2&0 stage 1 address translation.

M	Meaning
0b0	When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, EL2 stage 1 address translation disabled. When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL2&0 stage 1 address translation disabled. See the SCTLR_EL2.I field for the behavior of instruction accesses to Normal memory.
0b1	When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, EL2 stage 1 address translation enabled. When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL2&0 stage 1 address translation enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing SCTLR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SCTLR_EL2 or SCTLR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to SCTLR_EL2 are masked by SCTLRMASK_EL2.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SCTLR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR_EL2;
```

```
MSR SCTLR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_SRMASK) then
        SCTLR_EL2 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL2()) OR (SCTLR_EL2 AND
EffectiveSCTLRMASK_EL2());
    else
        SCTLR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, SCTLR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x110];
    else
        X[t, 64] = SCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLR_EL2;
    else
        X[t, 64] = SCTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR_EL1;
```

When FEAT_VHE is implemented

```
MSR SCTLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x110] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL1 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL1()) OR (SCTLR_EL1 AND
EffectiveSCTLRMASK_EL1());
        else
            SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL2 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL2()) OR (SCTLR_EL2 AND
EffectiveSCTLRMASK_EL2());
        else
            SCTLR_EL2 = X[t, 64];
    else
        SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR_EL3, System Control Register (EL3)

The SCTLR_EL3 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL3.

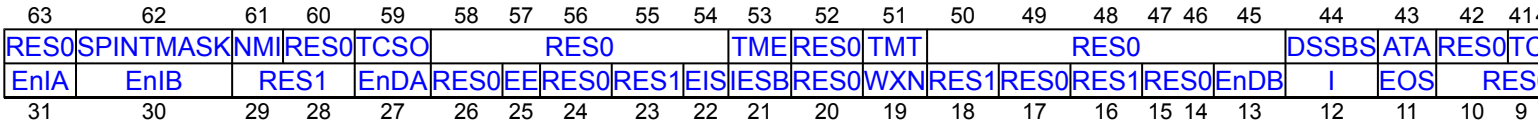
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR_EL3 are UNDEFINED.

Attributes

SCTLR_EL3 is a 64-bit register.

Field descriptions



Bit [63]

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

SP Interrupt Mask enable. When SCTLR_EL3.NMI is 1, controls whether PSTATE.SP acts as an interrupt mask, and controls the value of PSTATE.ALLINT on taking an exception to EL3.

SPINTMASK	Meaning
0b0	Does not cause PSTATE.SP to mask interrupts. PSTATE.ALLINT is set to 1 on taking an exception to EL3.
0b1	When PSTATE.SP is 1 and execution is at EL3, an IRQ or FIQ interrupt that is targeted to EL3 is masked regardless of any denotion of Superpriority. PSTATE.ALLINT is set to 0 on taking an exception to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]

When FEAT_NMI is implemented:

Non-maskable Interrupt enable.

NMI	Meaning
0b0	This control does not affect interrupt masking behavior.
0b1	This control enables all of the following: • The use of the PSTATE.ALLINT interrupt mask. • IRQ and FIQ interrupts to have Superpriority as an additional attribute. • PSTATE.SP to be used as an interrupt mask.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [60]

Reserved, RES0.

TCSO, bit [59]

When FEAT_MTE_STORE_ONLY is implemented:

Tag Checking Store Only.

TCSO	Meaning
0b0	This field has no effect on Tag checking.
0b1	Load instructions executed in EL3 are Tag Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [58:54]

Reserved, RES0.

TME, bit [53]

When FEAT_TME is implemented:

Enables the Transactional Memory Extension at EL3.

TME	Meaning
0b0	Any attempt to execute a TSTART instruction at EL3 is trapped, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL3.
0b1	This control does not cause any TSTART instruction to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [52]

Reserved, RES0.

TMT, bit [51]

When FEAT_TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL3.

TMT	Meaning
0b0	This control does not cause any TSTART instruction to fail.
0b1	When the TSTART instruction is executed at EL3, the transaction fails with a TRIVIAL failure cause.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [50:45]

Reserved, RES0.

DSSBS, bit [44]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to EL3.
0b1	PSTATE.SSBS is set to 1 on an exception to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL3.

Controls access to Allocation Tags and Tag Check operations in EL3.

ATA	Meaning
0b0	Access to Allocation Tags is prevented at EL3. Memory accesses at EL3 are not subject to a Tag Check operation.
0b1	This control does not prevent access to Allocation Tags at EL3. Tag Checked memory accesses at EL3 are subject to a Tag Check operation. The Tag Check operation depends on the type of tag at the memory being accessed: • For Allocation Tagged memory, an Allocation Tag Check operation. • If FEAT_MTE_CANONICAL_TAGS is implemented, for Canonically Tagged memory, a Canonical Tag Check operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

TCF, bits [41:40]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL3. Controls the effect of Tag Check Faults due to Loads and Stores in EL3.

TCF	Meaning	Applies when
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	When FEAT_MTE3 is implemented

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [39:38]

Reserved, RES0.

ITFSB, bit [37]

When FEAT_MTE_ASYNC is implemented:

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL3, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into TFSRE0_EL1 and TFSR_ELx registers.

ITFSB	Meaning
0b0	Tag Check Faults are not synchronized on entry to EL3.
0b1	Tag Check Faults are synchronized on entry to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bit [36]

When FEAT_BTI is implemented:

Indicates the Branch Type compatibility of the implicit BTI behavior for the following instructions at EL3:

- PACIASP.
- PACIBSP.
- If FEAT_PAuth_LR is implemented, PACIASPPC.
- If FEAT_PAuth_LR is implemented, PACIBSPPC.

BT	Meaning
0b0	When the PE is executing at EL3, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.jc.
0b1	When the PE is executing at EL3, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.c.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:32]

Reserved, RES0.

EnIA, bit [31]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIAKey_EL1 key, in the EL3 translation regime.

Possible values of this bit are:

EnIA	Meaning
0b0	Pointer authentication of instruction addresses, using the APIAKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses, using the APIAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIBKey_EL1 key, in the EL3 translation regime.

Possible values of this bit are:

EnIB	Meaning
0b0	Pointer authentication of instruction addresses, using the APIBKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses, using the APIBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:28]

Reserved, RES1.

EnDA, bit [27]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APDAKey_EL1 key, in the EL3 translation regime.

EnDA	Meaning
0b0	Pointer authentication of data addresses, using the APDAKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses, using the APDAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [26]

Reserved, RES0.

EE, bit [25]

When FEAT_MixedEnd is implemented:

Endianness of data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are little-endian.
0b1	Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are big-endian.

The EE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Endianness of data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are little-endian.
0b1	Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are big-endian.

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bit [24]

Reserved, RES0.

Bit [23]

Reserved, RES1.

EIS, bit [22]

When FEAT_ExS is implemented:

Exception Entry is Context Synchronizing.

EIS	Meaning
0b0	The taking of an exception to EL3 is not a context synchronizing event.
0b1	The taking of an exception to EL3 is a context synchronizing event.

If SCTLR_EL3.EIS is set to 0b0:

- Indirect writes to ESR_EL3, FAR_EL3, SPSR_EL3, ELR_EL3 are synchronized on exception entry to EL3, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.
- Some exception entries reported are not considered IFBEs per the memory model.

The following are not affected by the value of SCTLR_EL3.EIS:

- Changes to the PSTATE information on entry to EL3.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- The memory model requirement for exception entry to generate an Instruction Fetch Barrier Effect for some exception entries. See Basic definitions for the list of exception entries.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]

When FEAT_IESB is implemented:

Implicit Error Synchronization event enable.

IESB	Meaning
0b0	Disabled.
0b1	An implicit error synchronization event is added: • At each exception taken to EL3. • Before the operational pseudocode of each ERET instruction executed at EL3.

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field and, if implemented, SCR_EL3.NMEA. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSX instruction taken to EL3 and before each DRPS instruction executed at EL3, in addition to the other cases where it is added.

When FEAT_DoubleFault is implemented, the PE is in Non-debug state, and the Effective value of SCR_EL3.NMEA is 1, this field is ignored and its Effective value is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL3 translation regime, this bit can force all memory regions that are writable to be treated as XN.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the EL3 translation regime is forced to XN for accesses from software executing at EL3.

This bit applies only when SCTLR_EL3.M bit is set.

The WXN bit is permitted to be cached in a TLB.

This field is RES0 if TCR_EL3.PIE is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [18]

Reserved, RES1.

Bit [17]

Reserved, RES0.

Bit [16]

Reserved, RES1.

Bits [15:14]

Reserved, RES0.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APDBKey_EL1 key, in the EL3 translation regime.

EnDB	Meaning
0b0	Pointer authentication of data addresses, using the APDBKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses, using the APDBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL3:

I	Meaning
0b0	All instruction access to Normal memory from EL3 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR_EL3.M is 0, instruction accesses from stage 1 of the EL3 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	This control has no effect on the Cacheability of instruction access to Normal memory from EL3. If the value of SCTLR_EL3.M is 0, instruction accesses from stage 1 of the EL3 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

This bit has no effect on the EL1&0, EL2, or EL2&0 translation regimes.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

EOS, bit [11]

When FEAT_ExS is implemented:

Exception Exit is Context Synchronizing.

EOS	Meaning
0b0	An exception return from EL3 is not a context synchronizing event
0b1	An exception return from EL3 is a context synchronizing event

If SCTLR_EL3.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL3.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL3 and ELR_EL3 on exception return is synchronized.
- If the PE enters Debug state before the first instruction after an Exception return from EL3 to Non-secure state, any pending Halting debug event completes execution.
- The GIC behavior that allocates interrupts to FIQ or IRQ changes simultaneously with leaving the EL3 Exception level.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bits [10:7]

Reserved, RES0.

nAA, bit [6]

When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL3 under certain conditions.

The following instructions generate an Alignment fault if all bytes being accessed are not within a single naturally aligned 16-byte quantity, for access:

- LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH.
- STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH.
- If FEAT_LRCPC3 is implemented, the post index versions of LDAPR and the pre index versions of STLR.
- If FEAT_LRCPC3 and Advanced SIMD and floating-point instructions are implemented, LDAPUR (SIMD&FP), LDAP1 (SIMD&FP), STLUR (SIMD&FP), and STL1 (SIMD&FP).

If FEAT_LRCPC3 is implemented, the following instructions generate an Alignment fault if all bytes being accessed for a single register are not within a single naturally aligned 16-byte quantity, for access:

- LDIAPP, STILP.

nAA	Meaning
0b0	Unaligned accesses by the specified instructions generate an Alignment fault.
0b1	Unaligned accesses by the specified instructions do not generate an Alignment fault.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [5:4]

Reserved, RES1.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL3 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Cacheability control, for data accesses.

C	Meaning
0b0	All data access to Normal memory from EL3, and all Normal memory accesses to the EL3 translation tables, are Non-cacheable for all levels of data and unified cache.
0b1	This control has no effect on the Cacheability of: • Data access to Normal memory from EL3. • Normal memory accesses to the EL3 translation tables.

This bit has no effect on the EL1&0, EL2, or EL2&0 translation regimes.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL3.

A	Meaning
0b0	Alignment fault checking is disabled when executing at EL3. Alignment checks on some instructions are not disabled by this control. For more information, see 'Alignment of data accesses'.
0b1	Alignment fault checking is enabled when executing at EL3. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL3 stage 1 address translation. Possible values of this bit are:

M	Meaning
0b0	EL3 stage 1 address translation disabled. See the SCTLR_EL3.I field for the behavior of instruction accesses to Normal memory.
0b1	EL3 stage 1 address translation enabled.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Accessing SCTLR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR_EL3;
```

```
MSR SCTLR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.SCTLR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCTLR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLRMASK_EL1, System Control Masking Register (EL1)

The SCTLRMASK_EL1 characteristics are:

Purpose

Mask register to prevent updates of fields in SCTLR_EL1 on writes to SCTLR_EL1 or SCTLRALIAS_EL1.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLRMASK_EL1 are UNDEFINED.

Attributes

SCTLRMASK_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46
TIDCP	SPINTMASK	NMI	EnTP2	TCSO	TCSO0	EPAN	EnALS	EnAS0	EnASR	TME	TME0	TMT	TMT0	RES0			TWEDE
EnIA	EnIB	LSMAOE	nTLSMD	EnDA	UCI	EE	E0E	SPAN	EIS	IESB	TSCXT	WXN	nTWE	RES0	nTWI	UCT	DZE
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14

TIDCP, bit [63]

When FEAT_TIDCP1 is implemented:

Mask bit for TIDCP.

TIDCP	Meaning
0b0	SCTLR_EL1.TIDCP is writeable.
0b1	SCTLR_EL1.TIDCP is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

Mask bit for SPINTMASK.

SPINTMASK	Meaning
0b0	SCTLR_EL1.SPINTMASK is writeable.
0b1	SCTLR_EL1.SPINTMASK is not writeable.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL1, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]
When FEAT_NMI is implemented:

Mask bit for NMI.

NMI	Meaning
0b0	SCTLR_EL1.NMI is writeable.
0b1	SCTLR_EL1.NMI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTP2, bit [60]
When FEAT_SME is implemented:

Mask bit for EnTP2.

EnTP2	Meaning
0b0	SCTLR_EL1.EnTP2 is writeable.
0b1	SCTLR_EL1.EnTP2 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO, bit [59]
When FEAT_MTE_STORE_ONLY is implemented:

Mask bit for TCSO.

TCSO	Meaning
0b0	SCTLR_EL1.TCSO is writeable.
0b1	SCTLR_EL1.TCSO is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO0, bit [58]
When FEAT_MTE_STORE_ONLY is implemented:

Mask bit for TCSO0.

TCSO0	Meaning
0b0	SCTLR_EL1.TCSO0 is writeable.
0b1	SCTLR_EL1.TCSO0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EPAN, bit [57]
When FEAT_PAN3 is implemented:

Mask bit for EPAN.

EPAN	Meaning
0b0	SCTLR_EL1.EPAN is writeable.
0b1	SCTLR_EL1.EPAN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnALS, bit [56]
When FEAT_LS64 is implemented:

Mask bit for EnALS.

EnALS	Meaning
0b0	SCTLR_EL1.EnALS is writeable.
0b1	SCTLR_EL1.EnALS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [55]
When FEAT_LS64_ACCDATA is implemented:

Mask bit for EnAS0.

EnAS0	Meaning
0b0	SCTLR_EL1.EnAS0 is writeable.
0b1	SCTLR_EL1.EnAS0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnASR, bit [54]
When FEAT_LS64_V is implemented:

Mask bit for EnASR.

EnASR	Meaning
0b0	SCTLR_EL1.EnASR is writeable.
0b1	SCTLR_EL1.EnASR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME, bit [53]
When FEAT_TME is implemented:

Mask bit for TME.

TME	Meaning
0b0	SCTLR_EL1.TME is writeable.
0b1	SCTLR_EL1.TME is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME0, bit [52]
When FEAT_TME is implemented:

Mask bit for TME0.

TME0	Meaning
0b0	SCTLR_EL1.TME0 is writeable.
0b1	SCTLR_EL1.TME0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT, bit [51]
When FEAT_TME is implemented:

Mask bit for TMT.

TMT	Meaning
0b0	SCTLR_EL1.TMT is writeable.
0b1	SCTLR_EL1.TMT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT0, bit [50]
When FEAT_TME is implemented:

Mask bit for TMT0.

TMT0	Meaning
0b0	SCTLR_EL1.TMT0 is writeable.
0b1	SCTLR_EL1.TMT0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [49:47]

Reserved, RES0.

TWEDEL, bit [46]
When FEAT_TWED is implemented:

Mask bit for TWEDEL.

TWEDEL	Meaning
0b0	SCTLR_EL1.TWEDEL is writeable.
0b1	SCTLR_EL1.TWEDEL is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [45]
When FEAT_TWED is implemented:

Mask bit for TWEDEn.

TWEDEn	Meaning
0b0	SCTLR_EL1.TWEDEn is writeable.
0b1	SCTLR_EL1.TWEDEn is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSSBS, bit [44]
When FEAT_SSBS is implemented:

Mask bit for DSSBS.

DSSBS	Meaning
0b0	SCTLR_EL1.DSSBS is writeable.
0b1	SCTLR_EL1.DSSBS is not writeable.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL1, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Mask bit for ATA.

ATA	Meaning
0b0	SCTLR_EL1.ATA is writeable.
0b1	SCTLR_EL1.ATA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA0, bit [42]

When FEAT_MTE2 is implemented:

Mask bit for ATA0.

ATA0	Meaning
0b0	SCTLR_EL1.ATA0 is writeable.
0b1	SCTLR_EL1.ATA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [41]

Reserved, RES0.

TCF, bit [40]

When FEAT_MTE2 is implemented:

Mask bit for TCF.

TCF	Meaning
0b0	SCTLR_EL1.TCF is writeable.
0b1	SCTLR_EL1.TCF is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [39]

Reserved, RES0.

TCF0, bit [38]

When FEAT_MTE2 is implemented:

Mask bit for TCF0.

TCF0	Meaning
0b0	SCTLR_EL1.TCF0 is writeable.
0b1	SCTLR_EL1.TCF0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]

When FEAT_MTE_ASYNC is implemented:

Mask bit for ITFSB.

ITFSB	Meaning
0b0	SCTLR_EL1.ITFSB is writeable.
0b1	SCTLR_EL1.ITFSB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT1, bit [36]

When FEAT_BTI is implemented:

Mask bit for BT1.

BT1	Meaning
0b0	SCTLR_EL1.BT1 is writeable.
0b1	SCTLR_EL1.BT1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]
When FEAT_BTI is implemented:

Mask bit for BT0.

BT0	Meaning
0b0	SCTLR_EL1.BT0 is writeable.
0b1	SCTLR_EL1.BT0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [34]
When FEAT_FPMR is implemented:

Mask bit for EnFPM.

EnFPM	Meaning
0b0	SCTLR_EL1.EnFPM is writeable.
0b1	SCTLR_EL1.EnFPM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MSCEn, bit [33]
When FEAT_MOPS is implemented:

Mask bit for MSCEn.

MSCEn	Meaning
0b0	SCTLR_EL1.MSCEn is writeable.
0b1	SCTLR_EL1.MSCEn is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [32]
When FEAT_CMOW is implemented:

Mask bit for CMOW.

CMOW	Meaning
0b0	SCTLR_EL1.CMOW is writeable.
0b1	SCTLR_EL1.CMOW is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIA, bit [31]
When FEAT_PAuth is implemented:

Mask bit for EnIA.

EnIA	Meaning
0b0	SCTLR_EL1.EnIA is writeable.
0b1	SCTLR_EL1.EnIA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]
When FEAT_PAuth is implemented:

Mask bit for EnIB.

EnIB	Meaning
0b0	SCTLR_EL1.EnIB is writeable.
0b1	SCTLR_EL1.EnIB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]

When FEAT_LSMAOC is implemented:

Mask bit for LSMAOE.

LSMAOE	Meaning
0b0	SCTLR_EL1.LSMAOE is writeable.
0b1	SCTLR_EL1.LSMAOE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTLSMD, bit [28]

When FEAT_LSMAOC is implemented:

Mask bit for nTLSMD.

nTLSMD	Meaning
0b0	SCTLR_EL1.nTLSMD is writeable.
0b1	SCTLR_EL1.nTLSMD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnDA, bit [27]

When FEAT_PAuth is implemented:

Mask bit for EnDA.

EnDA	Meaning
0b0	SCTLR_EL1.EnDA is writeable.
0b1	SCTLR_EL1.EnDA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]

Mask bit for UCI.

UCI	Meaning
0b0	SCTLR_EL1.UCI is writeable.
0b1	SCTLR_EL1.UCI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EE, bit [25]

When FEAT_MixedEnd is implemented:

Mask bit for EE.

EE	Meaning
0b0	SCTLR_EL1.EE is writeable.
0b1	SCTLR_EL1.EE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0E, bit [24]

When FEAT_MixedEndEL0 is implemented:

Mask bit for E0E.

E0E	Meaning
0b0	SCTLR_EL1.E0E is writeable.
0b1	SCTLR_EL1.E0E is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPAN, bit [23]
When FEAT_PAN is implemented:

Mask bit for SPAN.

SPAN	Meaning
0b0	SCTLR_EL1.SPAN is writeable.
0b1	SCTLR_EL1.SPAN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EIS, bit [22]
When FEAT_ExS is implemented:

Mask bit for EIS.

EIS	Meaning
0b0	SCTLR_EL1.EIS is writeable.
0b1	SCTLR_EL1.EIS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IESB, bit [21]
When FEAT_IESB is implemented:

Mask bit for IESB.

IESB	Meaning
0b0	SCTLR_EL1.IESB is writeable.
0b1	SCTLR_EL1.IESB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSCXT, bit [20]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Mask bit for TSCXT.

TSCXT	Meaning
0b0	SCTLR_EL1.TSCXT is writeable.
0b1	SCTLR_EL1.TSCXT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WXN, bit [19]

Mask bit for WXN.

WXN	Meaning
0b0	SCTLR_EL1.WXN is writeable.
0b1	SCTLR_EL1.WXN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

Mask bit for nTWE.

nTWE	Meaning
0b0	SCTLR_EL1.nTWE is writeable.
0b1	SCTLR_EL1.nTWE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Mask bit for nTWI.

nTWI	Meaning
0b0	SCTLR_EL1.nTWI is writeable.
0b1	SCTLR_EL1.nTWI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

UCT, bit [15]

Mask bit for UCT.

UCT	Meaning
0b0	SCTLR_EL1.UCT is writeable.
0b1	SCTLR_EL1.UCT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DZE, bit [14]

Mask bit for DZE.

DZE	Meaning
0b0	SCTLR_EL1.DZE is writeable.
0b1	SCTLR_EL1.DZE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Mask bit for EnDB.

EnDB	Meaning
0b0	SCTLR_EL1.EnDB is writeable.
0b1	SCTLR_EL1.EnDB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Mask bit for I.

I	Meaning
0b0	SCTLR_EL1.I is writeable.
0b1	SCTLR_EL1.I is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EOS, bit [11]

When FEAT_ExS is implemented:

Mask bit for EOS.

EOS	Meaning
0b0	SCTLR_EL1.EOS is writeable.
0b1	SCTLR_EL1.EOS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnRCTX, bit [10]

When FEAT_SPECRES is implemented:

Mask bit for EnRCTX.

EnRCTX	Meaning
0b0	SCTLR_EL1.EnRCTX is writeable.
0b1	SCTLR_EL1.EnRCTX is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UMA, bit [9]

Mask bit for UMA.

UMA	Meaning
0b0	SCTLR_EL1.UMA is writeable.
0b1	SCTLR_EL1.UMA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SED, bit [8]

When FEAT_AA32EL0 is implemented:

Mask bit for SED.

SED	Meaning
0b0	SCTLR_EL1.SED is writeable.
0b1	SCTLR_EL1.SED is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITD, bit [7]

When FEAT_AA32EL0 is implemented:

Mask bit for ITD.

ITD	Meaning
0b0	SCTLR_EL1.ITD is writeable.
0b1	SCTLR_EL1.ITD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nAA, bit [6]

When FEAT_LSE2:

Mask bit for nAA.

nAA	Meaning
0b0	SCTLR_EL1.nAA is writeable.
0b1	SCTLR_EL1.nAA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CP15BEN, bit [5]

When FEAT_AA32EL0 is implemented:

Mask bit for CP15BEN.

CP15BEN	Meaning
0b0	SCTLR_EL1.CP15BEN is writeable.
0b1	SCTLR_EL1.CP15BEN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SA0, bit [4]

Mask bit for SA0.

SA0	Meaning
0b0	SCTLR_EL1.SA0 is writeable.
0b1	SCTLR_EL1.SA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SA, bit [3]

Mask bit for SA.

SA	Meaning
0b0	SCTLR_EL1.SA is writeable.
0b1	SCTLR_EL1.SA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Mask bit for C.

C	Meaning
0b0	SCTLR_EL1.C is writeable.
0b1	SCTLR_EL1.C is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A, bit [1]

Mask bit for A.

A	Meaning
0b0	SCTLR_EL1.A is writeable.
0b1	SCTLR_EL1.A is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

M, bit [0]

Mask bit for M.

M	Meaning
0b0	SCTLR_EL1.M is writeable.
0b1	SCTLR_EL1.M is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing SCTLRMASK_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name SCTLRMASK_EL1 or SCTLRMASK_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLRMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nSCTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x318];
    else
        X[t, 64] = SCTLRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLRMASK_EL2;
    else
        X[t, 64] = SCTLRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLRMASK_EL1;
```

```
MSR SCTLRMASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nSCTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x318] = X[t, 64];
    elsif !IsZero(EffectiveSCTLRMASK_EL1()) then
        UNDEFINED;
    else
        SCTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveSCTLRMASK_EL2()) then
            UNDEFINED;
        else
            SCTLRMASK_EL2 = X[t, 64];
    else
        SCTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLRMASK_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, SCTLRMASK_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x318];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SCTLRMASK_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLRMASK_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR SCTLRMASK_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x318] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SCTLRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SCTLRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLRMASK_EL2, System Control Masking Register (EL2)

The SCTLRMASK_EL2 characteristics are:

Purpose

Mask register to prevent updates of fields in SCTLR_EL2 on writes.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLRMASK_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

SCTLRMASK_EL2 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46
TIDCP	SPINTMASK	NMI	EnTP2	TCSO	TCSO0	EPAN	EnALS	EnAS0	EnASR	TME	TME0	TMT	TMT0	RES0			TWEDE
EnIA	EnIB	LSMAOE	nTLSMD	EnDA	UCI	EE	E0E	SPAN	EIS	IESB	TSCXT	WXN	nTWE	RES0	nTWI	UCT	DZE
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14

TIDCP, bit [63]

When FEAT_TIDCP1 is implemented:

Mask bit for TIDCP.

TIDCP	Meaning
0b0	SCTLR_EL2.TIDCP is writeable.
0b1	SCTLR_EL2.TIDCP is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

Mask bit for SPINTMASK.

SPINTMASK	Meaning
0b0	SCTLR_EL2.SPINTMASK is writeable.
0b1	SCTLR_EL2.SPINTMASK is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]
When FEAT_NMI is implemented:

Mask bit for NMI.

NMI	Meaning
0b0	SCTLR_EL2.NMI is writeable.
0b1	SCTLR_EL2.NMI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTP2, bit [60]
When FEAT_SME is implemented:

Mask bit for EnTP2.

EnTP2	Meaning
0b0	SCTLR_EL2.EnTP2 is writeable.
0b1	SCTLR_EL2.EnTP2 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO, bit [59]
When FEAT_MTE_STORE_ONLY is implemented:

Mask bit for TCSO.

TCSO	Meaning
0b0	SCTLR_EL2.TCSO is writeable.
0b1	SCTLR_EL2.TCSO is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO0, bit [58]
When FEAT_MTE_STORE_ONLY is implemented:

Mask bit for TCSO0.

TCSO0	Meaning
0b0	SCTLR_EL2.TCSO0 is writeable.
0b1	SCTLR_EL2.TCSO0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EPAN, bit [57]
When FEAT_PAN3 is implemented:

Mask bit for EPAN.

EPAN	Meaning
0b0	SCTLR_EL2.EPAN is writeable.
0b1	SCTLR_EL2.EPAN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnALS, bit [56]
When FEAT_LS64 is implemented:

Mask bit for EnALS.

EnALS	Meaning
0b0	SCTLR_EL2.EnALS is writeable.
0b1	SCTLR_EL2.EnALS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [55]
When FEAT_LS64_ACCDATA is implemented:

Mask bit for EnAS0.

EnAS0	Meaning
0b0	SCTLR_EL2.EnAS0 is writeable.
0b1	SCTLR_EL2.EnAS0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnASR, bit [54]
When FEAT_LS64_V is implemented:

Mask bit for EnASR.

EnASR	Meaning
0b0	SCTLR_EL2.EnASR is writeable.
0b1	SCTLR_EL2.EnASR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME, bit [53]
When FEAT_TME is implemented:

Mask bit for TME.

TME	Meaning
0b0	SCTLR_EL2.TME is writeable.
0b1	SCTLR_EL2.TME is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME0, bit [52]

When FEAT_TME is implemented:

Mask bit for TME0.

TME0	Meaning
0b0	SCTLR_EL2.TME0 is writeable.
0b1	SCTLR_EL2.TME0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT, bit [51]

When FEAT_TME is implemented:

Mask bit for TMT.

TMT	Meaning
0b0	SCTLR_EL2.TMT is writeable.
0b1	SCTLR_EL2.TMT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT0, bit [50]

When FEAT_TME is implemented:

Mask bit for TMT0.

TMT0	Meaning
0b0	SCTLR_EL2.TMT0 is writeable.
0b1	SCTLR_EL2.TMT0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [49:47]

Reserved, RES0.

TWEDEL, bit [46]

When FEAT_TWED is implemented:

Mask bit for TWEDEL.

TWEDEL	Meaning
0b0	SCTLR_EL2.TWEDEL is writeable.
0b1	SCTLR_EL2.TWEDEL is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [45]

When FEAT_TWED is implemented:

Mask bit for TWEDEn.

TWEDEn	Meaning
0b0	SCTLR_EL2.TWEDEn is writeable.
0b1	SCTLR_EL2.TWEDEn is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSSBS, bit [44]

When FEAT_SSBS is implemented:

Mask bit for DSSBS.

DSSBS	Meaning
0b0	SCTLR_EL2.DSSBS is writeable.
0b1	SCTLR_EL2.DSSBS is not writeable.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Mask bit for ATA.

ATA	Meaning
0b0	SCTLR_EL2.ATA is writeable.
0b1	SCTLR_EL2.ATA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA0, bit [42]

When FEAT_MTE2 is implemented:

Mask bit for ATA0.

ATA0	Meaning
0b0	SCTLR_EL2.ATA0 is writeable.
0b1	SCTLR_EL2.ATA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [41]

Reserved, RES0.

TCF, bit [40]

When FEAT_MTE2 is implemented:

Mask bit for TCF.

TCF	Meaning
0b0	SCTLR_EL2.TCF is writeable.
0b1	SCTLR_EL2.TCF is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [39]

Reserved, RES0.

TCF0, bit [38]

When FEAT_MTE2 is implemented:

Mask bit for TCF0.

TCF0	Meaning
0b0	SCTLR_EL2.TCF0 is writeable.
0b1	SCTLR_EL2.TCF0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]

When FEAT_MTE_ASYNC is implemented:

Mask bit for ITFSB.

ITFSB	Meaning
0b0	SCTLR_EL2.ITFSB is writeable.
0b1	SCTLR_EL2.ITFSB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bit [36]

When FEAT_BTI is implemented:

Mask bit for BT.

BT	Meaning
0b0	SCTLR_EL2.BT is writeable.
0b1	SCTLR_EL2.BT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]

When FEAT_BTI is implemented:

Mask bit for BT0.

BT0	Meaning
0b0	SCTLR_EL2.BT0 is writeable.
0b1	SCTLR_EL2.BT0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [34]

When FEAT_FPMR is implemented:

Mask bit for EnFPM.

EnFPM	Meaning
0b0	SCTLR_EL2.EnFPM is writeable.
0b1	SCTLR_EL2.EnFPM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MSCEn, bit [33]

When FEAT_MOPS is implemented:

Mask bit for MSCEn.

MSCEn	Meaning
0b0	SCTLR_EL2.MSCEn is writeable.
0b1	SCTLR_EL2.MSCEn is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [32]
When FEAT_CMOW is implemented:

Mask bit for CMOW.

CMOW	Meaning
0b0	SCTLR_EL2.CMOW is writeable.
0b1	SCTLR_EL2.CMOW is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIA, bit [31]
When FEAT_PAuth is implemented:

Mask bit for EnIA.

EnIA	Meaning
0b0	SCTLR_EL2.EnIA is writeable.
0b1	SCTLR_EL2.EnIA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]
When FEAT_PAuth is implemented:

Mask bit for EnIB.

EnIB	Meaning
0b0	SCTLR_EL2.EnIB is writeable.
0b1	SCTLR_EL2.EnIB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]
When FEAT_LSMAOC is implemented:

Mask bit for LSMAOE.

LSMAOE	Meaning
0b0	SCTLR_EL2.LSMAOE is writeable.
0b1	SCTLR_EL2.LSMAOE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTLSMD, bit [28]
When FEAT_LSMAOC is implemented:

Mask bit for nTLSMD.

nTLSMD	Meaning
0b0	SCTLR_EL2.nTLSMD is writeable.
0b1	SCTLR_EL2.nTLSMD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnDA, bit [27]
When FEAT_PAuth is implemented:

Mask bit for EnDA.

EnDA	Meaning
0b0	SCTLR_EL2.EnDA is writeable.
0b1	SCTLR_EL2.EnDA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]

Mask bit for UCI.

UCI	Meaning
0b0	SCTLR_EL2.UCI is writeable.
0b1	SCTLR_EL2.UCI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EE, bit [25]

When FEAT_MixedEnd is implemented:

Mask bit for EE.

EE	Meaning
0b0	SCTLR_EL2.EE is writeable.
0b1	SCTLR_EL2.EE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0E, bit [24]

When FEAT_MixedEndEL0 is implemented:

Mask bit for E0E.

E0E	Meaning
0b0	SCTLR_EL2.E0E is writeable.
0b1	SCTLR_EL2.E0E is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPAN, bit [23]

Mask bit for SPAN.

SPAN	Meaning
0b0	SCTLR_EL2.SPAN is writeable.
0b1	SCTLR_EL2.SPAN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EIS, bit [22]

When FEAT_ExS is implemented:

Mask bit for EIS.

EIS	Meaning
0b0	SCTLR_EL2.EIS is writeable.
0b1	SCTLR_EL2.EIS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IESB, bit [21]

When FEAT_IESB is implemented:

Mask bit for IESB.

IESB	Meaning
0b0	SCTLR_EL2.IESB is writeable.
0b1	SCTLR_EL2.IESB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSCXT, bit [20]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Mask bit for TSCXT.

TSCXT	Meaning
0b0	SCTLR_EL2.TSCXT is writeable.
0b1	SCTLR_EL2.TSCXT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WXN, bit [19]

Mask bit for WXN.

WXN	Meaning
0b0	SCTLR_EL2.WXN is writeable.
0b1	SCTLR_EL2.WXN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

Mask bit for nTWE.

nTWE	Meaning
0b0	SCTLR_EL2.nTWE is writeable.
0b1	SCTLR_EL2.nTWE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Mask bit for nTWI.

nTWI	Meaning
0b0	SCTLR_EL2.nTWI is writeable.
0b1	SCTLR_EL2.nTWI is not writeable.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

UCT, bit [15]

Mask bit for UCT.

UCT	Meaning
0b0	SCTLR_EL2.UCT is writeable.
0b1	SCTLR_EL2.UCT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DZE, bit [14]

Mask bit for DZE.

DZE	Meaning
0b0	SCTLR_EL2.DZE is writeable.
0b1	SCTLR_EL2.DZE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Mask bit for EnDB.

EnDB	Meaning
0b0	SCTLR_EL2.EnDB is writeable.
0b1	SCTLR_EL2.EnDB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Mask bit for I.

I	Meaning
0b0	SCTLR_EL2.I is writeable.
0b1	SCTLR_EL2.I is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EOS, bit [11]

When FEAT_ExS is implemented:

Mask bit for EOS.

EOS	Meaning
0b0	SCTLR_EL2.EOS is writeable.
0b1	SCTLR_EL2.EOS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnRCTX, bit [10]

When FEAT_SPECRES is implemented:

Mask bit for EnRCTX.

EnRCTX	Meaning
0b0	SCTLR_EL2.EnRCTX is writeable.
0b1	SCTLR_EL2.EnRCTX is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [9]

Reserved, RES0.

SED, bit [8]

When FEAT_AA32EL0 is implemented:

Mask bit for SED.

SED	Meaning
0b0	SCTLR_EL2.SED is writeable.
0b1	SCTLR_EL2.SED is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITD, bit [7]

When FEAT_AA32EL0 is implemented:

Mask bit for ITD.

ITD	Meaning
0b0	SCTLR_EL2.ITD is writeable.
0b1	SCTLR_EL2.ITD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nAA, bit [6]

Mask bit for nAA.

nAA	Meaning
0b0	SCTLR_EL2.nAA is writeable.
0b1	SCTLR_EL2.nAA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CP15BEN, bit [5]

When FEAT_AA32EL0 is implemented:

Mask bit for CP15BEN.

CP15BEN	Meaning
0b0	SCTLR_EL2.CP15BEN is writeable.
0b1	SCTLR_EL2.CP15BEN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SA0, bit [4]

Mask bit for SA0.

SA0	Meaning
0b0	SCTLR_EL2.SA0 is writeable.
0b1	SCTLR_EL2.SA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SA, bit [3]

Mask bit for SA.

SA	Meaning
0b0	SCTLR_EL2.SA is writeable.
0b1	SCTLR_EL2.SA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Mask bit for C.

C	Meaning
0b0	SCTLR_EL2.C is writeable.
0b1	SCTLR_EL2.C is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A, bit [1]

Mask bit for A.

A	Meaning
0b0	SCTLR_EL2.A is writeable.
0b1	SCTLR_EL2.A is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

M, bit [0]

Mask bit for M.

M	Meaning
0b0	SCTLR_EL2.M is writeable.
0b1	SCTLR_EL2.M is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing SCTLRMASK_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SCTLRMASK_EL2 or SCTLRMASK_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLRMASK_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCTLRMASK_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLRMASK_EL2;
```

```
MSR SCTLRMASK_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsZero(EffectiveSCTLRMASK_EL2()) then
        UNDEFINED;
    else
        SCTLRMASK_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLRMASK_EL2 = X[t, 64];
```

MRS <Xt>, SCTLRMASK_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nSCTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x318];
    else
        X[t, 64] = SCTLRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLRMASK_EL2;
    else
        X[t, 64] = SCTLRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLRMASK_EL1;
```

MSR SCTLRMASK_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nSCTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x318] = X[t, 64];
    elsif !IsZero(EffectiveSCTLRMASK_EL1()) then
        UNDEFINED;
    else
        SCTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveSCTLRMASK_EL2()) then
            UNDEFINED;
        else
            SCTLRMASK_EL2 = X[t, 64];
    else
        SCTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLRMASK_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCXTNUM_EL0, EL0 Read/Write Software Context Number

The SCXTNUM_EL0 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL0 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

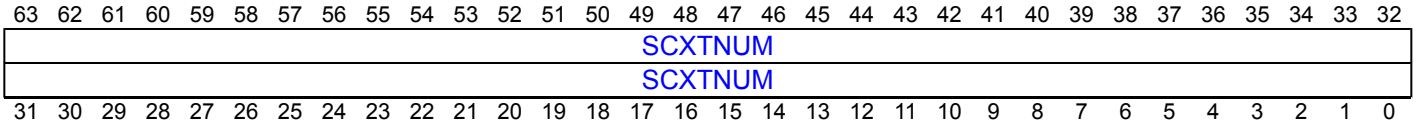
Configuration

This register is present only when (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to SCXTNUM_EL0 are UNDEFINED.

Attributes

SCXTNUM_EL0 is a 64-bit register.

Field descriptions



SCXTNUM, bits [63:0]

Software Context Number. A number to identify the context within the EL0 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCXTNUM_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.TSCXT == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGRTR_EL2.SCXTNUM_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TSCXT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCXTNUM_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.SCXTNUM_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCXTNUM_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCXTNUM_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCXTNUM_EL0;
```

```
MSR SCXTNUM_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.TSCXT == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCXTNUM_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TSCXT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCXTNUM_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.SCXTNUM_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCXTNUM_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCXTNUM_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL0 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCXTNUM_EL1, EL1 Read/Write Software Context Number

The SCXTNUM_EL1 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL1 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

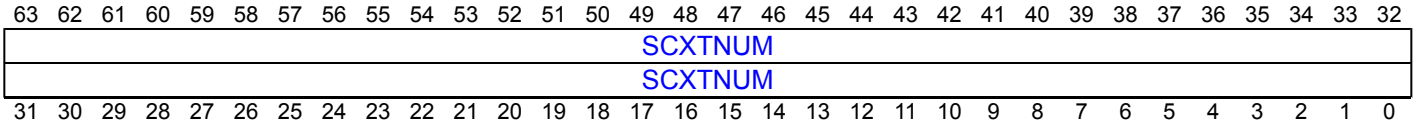
Configuration

This register is present only when (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to SCXTNUM_EL1 are UNDEFINED.

Attributes

SCXTNUM_EL1 is a 64-bit register.

Field descriptions



SCXTNUM, bits [63:0]

Software Context Number. A number to identify the context within the EL1 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name SCXTNUM_EL1 or SCXTNUM_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCXTNUM_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.SCXTNUM_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x188];
    else
        X[t, 64] = SCXTNUM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCXTNUM_EL2;
    else
        X[t, 64] = SCXTNUM_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCXTNUM_EL1;
```

```
MSR SCXTNUM_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.SCXTNUM_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x188] = X[t, 64];
    else
        SCXTNUM_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        SCXTNUM_EL2 = X[t, 64];
    else
        SCXTNUM_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, SCXTNUM_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x188];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SCXTNUM_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCXTNUM_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR SCXTNUM_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x188] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SCXTNUM_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SCXTNUM_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCXTNUM_EL2, EL2 Read/Write Software Context Number

The SCXTNUM_EL2 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL2 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

Configuration

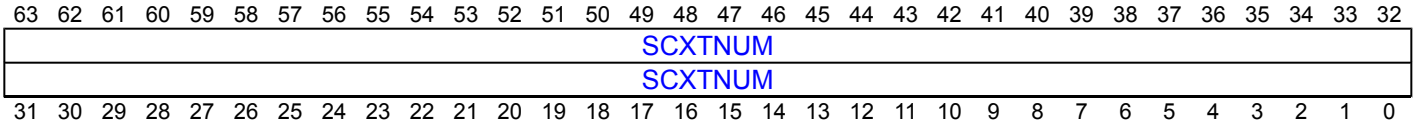
This register is present only when (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to SCXTNUM_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

SCXTNUM_EL2 is a 64-bit register.

Field descriptions



SCXTNUM, bits [63:0]

Software Context Number. A number to identify the context within the EL2 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SCXTNUM_EL2 or SCXTNUM_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCXTNUM_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCXTNUM_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCXTNUM_EL2;
```

MSR SCXTNUM_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCXTNUM_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL2 = X[t, 64];
```

MRS <Xt>, SCXTNUM_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.SCXTNUM_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x188];
    else
        X[t, 64] = SCXTNUM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCXTNUM_EL2;
    else
        X[t, 64] = SCXTNUM_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCXTNUM_EL1;
```

```
MSR SCXTNUM_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.SCXTNUM_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x188] = X[t, 64];
    else
        SCXTNUM_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        SCXTNUM_EL2 = X[t, 64];
    else
        SCXTNUM_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCXTNUM_EL3, EL3 Read/Write Software Context Number

The SCXTNUM_EL3 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL3 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

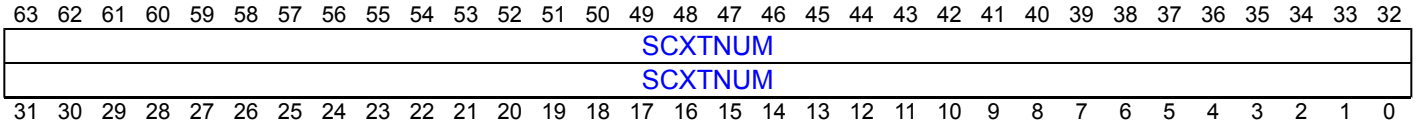
Configuration

This register is present only when EL3 is implemented, (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented), and FEAT_AA64 is implemented. Otherwise, direct accesses to SCXTNUM_EL3 are UNDEFINED.

Attributes

SCXTNUM_EL3 is a 64-bit register.

Field descriptions



SCXTNUM, bits [63:0]

Software Context Number. A number to identify the context within the EL3 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCXTNUM_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1101	0b0000	0b111

```
if !(HaveEL(EL3) && (IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCXTNUM_EL3;
```

```
MSR SCXTNUM_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1101	0b0000	0b111

```
if !(HaveEL(EL3) && (IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SDER32_EL2, AArch32 Secure Debug Enable Register

The SDER32_EL2 characteristics are:

Purpose

Allows access to the AArch32 register SDER from Secure EL2 and EL3 only.

Configuration

AArch64 System register SDER32_EL2 bits [63:0] are architecturally mapped to AArch64 System register SDER32_EL3[63:0] when EL3 is implemented.

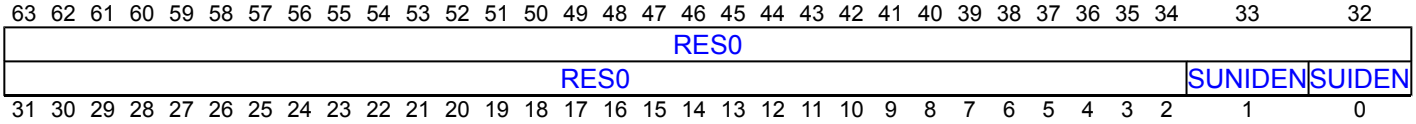
AArch64 System register SDER32_EL2 bits [31:0] are architecturally mapped to AArch32 System register SDER[31:0].

This register is present only when EL2 is implemented, FEAT_SEL2 is implemented, FEAT_AA32EL1 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to SDER32_EL2 are UNDEFINED.

Attributes

SDER32_EL2 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

SUNIDEN, bit [1]

Secure User Non-Invasive Debug Enable.

SUNIDEN	Meaning
0b0	This bit has no effect on non-invasive debug.
0b1	Non-invasive debug is allowed in Secure EL0 using AArch32.

When Secure EL1 is using AArch32, the forms of non-invasive debug affected by this control are:

- The PC Sample-based Profiling Extension. See About the PC Sample-based Profiling Extension.
- When SelfHostedTraceEnabled() == FALSE, processor trace.
- When EL3 is implemented, Performance Monitors.

When Secure EL1 is using AArch64, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SUIDEN, bit [0]

When EL3 is implemented:

Secure User Invasive Debug Enable.

SUIDEN	Meaning
0b0	This bit does not affect the generation of debug exceptions at Secure EL0.
0b1	If EL1 is using AArch32, debug exceptions from Secure EL0 are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing SDER32_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SDER32_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0011	0b001

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA32EL1) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !HaveEL(EL2) || !IsFeatureImplemented(FEAT_SEL2) || !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SDER32_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = SDER32_EL2;
```

MSR SDER32_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0011	0b001

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA32EL1) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !HaveEL(EL2) || !IsFeatureImplemented(FEAT_SEL2) || !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SDER32_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        SDER32_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SDER32_EL3, AArch32 Secure Debug Enable Register

The SDER32_EL3 characteristics are:

Purpose

Allows access to the AArch32 register SDER from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register SDER32_EL3 bits [63:0] are architecturally mapped to AArch64 System register SDER32_EL2[63:0] when EL2 is implemented and FEAT_SEL2 is implemented.

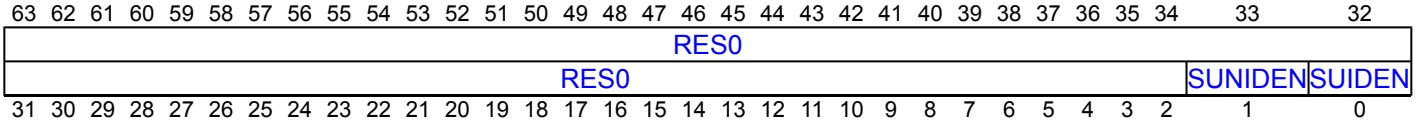
AArch64 System register SDER32_EL3 bits [31:0] are architecturally mapped to AArch32 System register SDER[31:0].

This register is present only when EL3 is implemented, FEAT_AA32EL1 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to SDER32_EL3 are UNDEFINED.

Attributes

SDER32_EL3 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

SUNIDEN, bit [1]

Secure User Non-Invasive Debug Enable.

SUNIDEN	Meaning
0b0	This bit has no effect on non-invasive debug.
0b1	Non-invasive debug is allowed in Secure EL0 using AArch32.

When Secure EL1 is using AArch32, the forms of non-invasive debug affected by this control are:

- The PC Sample-based Profiling Extension. See About the PC Sample-based Profiling Extension.
- When SelfHostedTraceEnabled() == FALSE, processor trace.
- Performance Monitors.

When Secure EL1 is using AArch64, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SUIDEN, bit [0]

Secure User Invasive Debug Enable.

SUIDEN	Meaning
0b0	This bit does not affect the generation of debug exceptions at Secure EL0.
0b1	If EL1 is using AArch32, debug exceptions from Secure EL0 are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SDER32_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SDER32_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !HaveEL(EL3) || !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SDER32_EL3;
```

```
MSR SDER32_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !HaveEL(EL3) || !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SDER32_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SMCR_EL1, SME Control Register (EL1)

The SMCR_EL1 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at Exception levels EL1 and EL0.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMCR_EL1 are UNDEFINED.

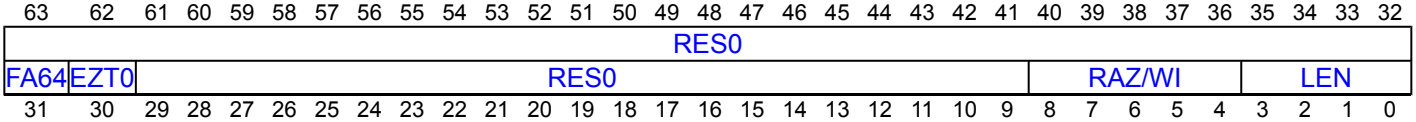
This register has no effect if the PE is not in Streaming SVE mode.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this register has no effect on execution at EL0.

Attributes

SMCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction at EL1 is considered legal when executed in Streaming SVE mode.

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, controls whether execution of an A64 instruction at EL0 is considered legal when executed in Streaming SVE mode.

FA64	Meaning
0b0	This control does not cause any instruction to be treated as legal when executed in Streaming SVE mode.
0b1	This control causes all implemented A64 instructions to be treated as legal when executed in Streaming SVE mode at EL1 and EL0, if they are treated as legal at more privileged Exception levels in the current Security state.

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EZT0, bit [30]

When FEAT_SME2 is implemented:

Traps execution at EL1 and EL0 of the LDR, LUTI2, LUTI4, MOVT, STR, and ZERO instructions that access the ZT0 register to EL1, or to EL2 when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1.

The exception is reported using ESR_EL1.EC or ESR_EL2.EC value `0x1D`, with an ISS code of `0x0000004`, at a lower priority than a trap due to PSTATE.SM or PSTATE.ZA.

EZT0	Meaning
`0b0`	This control causes execution of these instructions at EL1 and EL0 to be trapped.
`0b1`	This control does not cause execution of any instruction to be trapped.

Changes to this field only affect whether instructions that access ZT0 are trapped. They do not affect the contents of ZT0, which remain valid so long as PSTATE.ZA is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Streaming SVE vector length (SVL) at EL1 of (LEN+1)*128 bits. This field also defines the Effective Streaming SVE vector length at EL0 when EL2 is not implemented, or EL2 is not enabled in the current Security state, or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

The Streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support any subset of the architecturally permitted lengths.

When the PE is in Streaming SVE mode, the Effective SVE vector length (VL) is equal to SVL.

When FEAT_SVE is implemented, and the PE is not in Streaming SVE mode, VL is equal to the Effective Non-streaming SVE vector length. See ZCR_EL1.

For all purposes other than returning the result of a direct read of SMCR_EL1, the PE selects the Effective Streaming SVE vector length by performing checks in the following order:

1. If the requested length is less than the minimum implemented Streaming SVE vector length, then the Effective length is the minimum implemented Streaming SVE vector length.
2. If EL2 is implemented and enabled in the current Security state, and the requested length is greater than the Effective length at EL2, then the Effective length at EL2 is used.
3. If EL3 is implemented and the requested length is greater than the Effective length at EL3, then the Effective length at EL3 is used.
4. Otherwise, the Effective length is the highest supported Streaming SVE vector length that is less than or equal to the requested length.

An indirect read of SMCR_EL1.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMCR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name SMCR_EL1 or SMCR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SMCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x1F0];
    else
        X[t, 64] = SMCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SMCR_EL2;
    else
        X[t, 64] = SMCR_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SMCR_EL1;
```

```
MSR SMCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x1F0] = X[t, 64];
    else
        SMCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif ELIsInHost(EL2) then
        SMCR_EL2 = X[t, 64];
    else
        SMCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, SMCR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x1F0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
            UNDEFINED;
        elsif CPTR_EL2.SMEN IN {'x0'} then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            X[t, 64] = SMCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CPTR_EL3.ESM == '0' then
            AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            X[t, 64] = SMCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR SMCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x1F0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
            UNDEFINED;
        elsif CPTR_EL2.SMEN IN {'x0'} then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            SMCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CPTR_EL3.ESM == '0' then
            AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            SMCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SMCR_EL2, SME Control Register (EL2)

The SMCR_EL2 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at Exception levels EL2, EL1, and EL0.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMCR_EL2 are UNDEFINED.

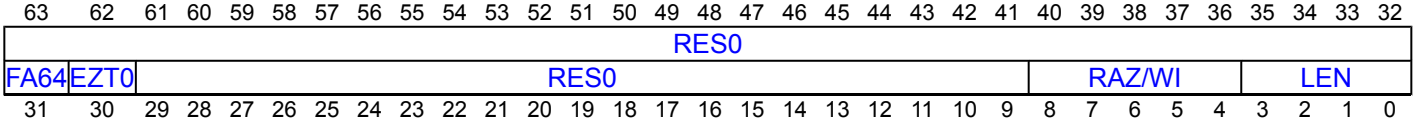
This register has no effect if the PE is not in Streaming SVE mode, or if EL2 is not enabled in the current Security state.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

SMCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction at EL2, EL1, and EL0 when EL2 is implemented and enabled in the current Security state is considered legal when executed in Streaming SVE mode.

FA64	Meaning
0b0	This control does not cause any instruction to be treated as legal when executed in Streaming SVE mode.
0b1	This control causes all implemented A64 instructions to be treated as legal when executed in Streaming SVE mode at EL2, EL1, and EL0, if they are treated as legal at EL3.

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EZT0, bit [30]

When FEAT_SME2 is implemented:

Traps execution at EL2, EL1, and EL0 of the LDR, LUTI2, LUTI4, MOVT, STR, and ZERO instructions that access the ZT0 register to EL2, when EL2 is enabled in the current Security state.

The exception is reported using ESR_EL2.EC value `0x1D`, with an ISS code of `0x0000004`, at a lower priority than a trap due to PSTATE.SM or PSTATE.ZA.

EZT0	Meaning
`0b0`	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
`0b1`	This control does not cause execution of any instruction to be trapped.

Changes to this field only affect whether instructions that access ZT0 are trapped. They do not affect the contents of ZT0, which remain valid so long as PSTATE.ZA is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Streaming SVE vector length (SVL) at EL2 of (LEN+1)*128 bits. This field also defines the Effective Streaming SVE vector length at EL0 when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support any subset of the architecturally permitted lengths.

When the PE is in Streaming SVE mode, the Effective SVE vector length (VL) is equal to SVL.

When FEAT_SVE is implemented, and the PE is not in Streaming SVE mode, VL is equal to the Effective Non-streaming SVE vector length. See ZCR_EL2.

For all purposes other than returning the result of a direct read of SMCR_EL2, the PE selects the Effective Streaming SVE vector length by performing checks in the following order:

1. If the requested length is less than the minimum implemented Streaming SVE vector length, then the Effective length is the minimum implemented Streaming SVE vector length.
2. If EL3 is implemented and the requested length is greater than the Effective length at EL3, then the Effective length at EL3 is used.
3. Otherwise, the Effective length is the highest supported Streaming SVE vector length that is less than or equal to the requested length.

An indirect read of SMCR_EL2.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMCR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SMCR_EL2 or SMCR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SMCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SMCR_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SMCR_EL2;
```

```
MSR SMCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL2 = X[t, 64];
```

MRS <Xt>, SMCR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x1F0];
    else
        X[t, 64] = SMCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SMCR_EL2;
    else
        X[t, 64] = SMCR_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SMCR_EL1;
```

```
MSR SMCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x1F0] = X[t, 64];
    else
        SMCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif ELIsInHost(EL2) then
        SMCR_EL2 = X[t, 64];
    else
        SMCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SMCR_EL3, SME Control Register (EL3)

The SMCR_EL3 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at all Exception levels.

Configuration

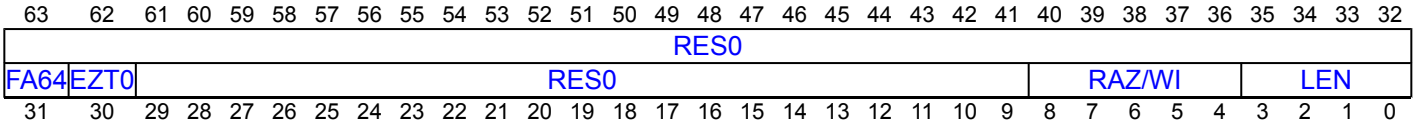
This register is present only when FEAT_SME is implemented and EL3 is implemented. Otherwise, direct accesses to SMCR_EL3 are UNDEFINED.

This register has no effect if the PE is not in Streaming SVE mode.

Attributes

SMCR_EL3 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction is considered legal when executed in Streaming SVE mode.

FA64	Meaning
0b0	This control does not cause any instruction to be treated as legal when executed in Streaming SVE mode.
0b1	This control causes all implemented A64 instructions to be treated as legal when executed in Streaming SVE mode .

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EZT0, bit [30]

When FEAT_SME2 is implemented:

Traps execution at all Exception levels of the LDR, LUTI2, LUTI4, MOVT, STR, and ZERO instructions that access the ZT0 register to EL3.

The exception is reported using ESR_EL3.EC value 0x1D, with an ISS code of 0x0000004, at a lower priority than a trap due to PSTATE.SM or PSTATE.ZA.

EZT0	Meaning
0b0	This control causes execution of these instructions at all Exception levels to be trapped.
0b1	This control does not cause execution of any instruction to be trapped.

Changes to this field only affect whether instructions that access ZT0 are trapped. They do not affect the contents of ZT0, which remain valid so long as PSTATE.ZA is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Streaming SVE vector length (SVL) at EL3 of (LEN+1)*128 bits.

The Streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support any subset of the architecturally permitted lengths.

When the PE is in Streaming SVE mode, the Effective SVE vector length (VL) is equal to SVL.

When FEAT_SVE is implemented, and the PE is not in Streaming SVE mode, VL is equal to the Effective Non-streaming SVE vector length. See ZCR_EL3.

For all purposes other than returning the result of a direct read of SMCR_EL3, the PE selects the Effective Streaming SVE vector length by performing checks in the following order:

1. If the requested length is less than the minimum implemented Streaming SVE vector length, then the Effective length is the minimum implemented Streaming SVE vector length.
2. Otherwise, the Effective length is the highest supported Streaming SVE vector length that is less than or equal to the requested length.

An indirect read of SMCR_EL3.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SMCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0010	0b110

```
if !(IsFeatureImplemented(FEAT_SME) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SMCR_EL3;
```

```
MSR SMCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0010	0b110

```
if !(IsFeatureImplemented(FEAT_SME) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SMIDR_EL1, Streaming Mode Identification Register

The SMIDR_EL1 characteristics are:

Purpose

Provides additional identification mechanisms for scheduling purposes, for a PE that supports Streaming SVE mode.

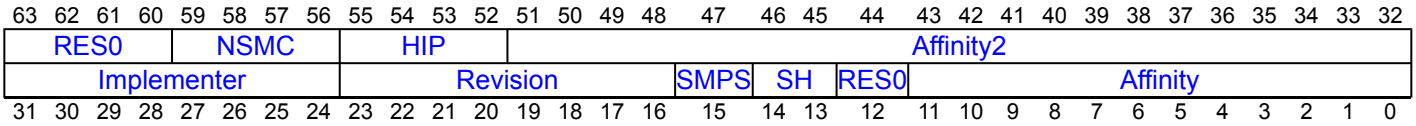
Configuration

This register is present only when FEAT_SME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SMIDR_EL1 are UNDEFINED.

Attributes

SMIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:60]

Reserved, RES0.

NSMC, bits [59:56]

If SMIDR_EL1.{SH,Affinity} indicates that the implementation of Streaming SVE mode is shared, then this field identifies the number of SMCUs, minus 1, associated with the concatenated SMIDR_EL1.{Affinity2,Affinity} 32-bit value.

If SMIDR_EL1.{SH,Affinity} indicates that the implementation of Streaming SVE mode is not shared, then this field is zero and should be ignored by software.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NSMC	Meaning
0b0000	The implementation of Streaming SVE mode associated with this PE is not shared or is a single SMCU.
0b0001..0b1110	The number of SMCUs in the group of SMCUs providing the implementation of Streaming SVE mode for this PE, minus 1.
0b1111	Reserved.

Access to this field is **RO**.

HIP, bits [55:52]

When FEAT_SME2p2 is implemented and SMIDR_EL1.SMPS == 1:

Highest Implemented Priority. If Streaming SVE mode execution priority is supported, this field indicates the range of priority levels implemented by the PE, and the Highest Implemented Priority value.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HIP	Meaning
0b0000	All streaming execution priorities from 0 to 15 are implemented. The Highest Implemented Priority value is 15.
0b0001..0b1111	All streaming execution priorities less than or equal to this value are implemented. The Highest Implemented Priority value is the value of this field.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Affinity2, bits [51:32]

The most significant 20 bits of the SMCU affinity for this PE, to be used in conjunction with SMIDR_EL1.Affinity.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Implementer	Meaning
0x00	Reserved for software use.
0x41	Arm Limited.
0x42	Broadcom Corporation.
0x43	Cavium Inc.
0x44	Digital Equipment Corporation.
0x46	Fujitsu Ltd.
0x49	Infineon Technologies AG.
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation.
0x50	Applied Micro Circuits Corporation.
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation.
0xC0	Ampere Computing.

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

It is not required that this value is the same as the value of MIDR_EL1.Implementer.

Access to this field is **RO**.

Revision, bits [23:16]

Revision number for the Streaming Mode Compute Unit (SMCU).

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SMPS, bit [15]

Indicates support for Streaming SVE mode execution priority.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SMPS	Meaning
0b0	Priority control not supported.
0b1	Priority control supported.

Access to this field is **RO**.

SH, bits [14:13]

Indicates whether the implementation of Streaming SVE mode in this PE is shared with other PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SH	Meaning
0b00	Refer to SMIDR_EL1.Affinity.
0b01	Reserved.
0b10	The implementation of Streaming SVE mode is not shared with other PEs.
0b11	The implementation of Streaming SVE mode is shared with other PEs.

Access to this field is **RO**.

Bit [12]

Reserved, RES0.

Affinity, bits [11:0]

The least significant 12 bits of the SMCU affinity for this PE.

If the implementation of Streaming SVE mode is shared, then the concatenated SMIDR_EL1.{Affinity2,Affinity} 32-bit value identifies which shared SMCUs are associated with this PE. Every PE that shares the same SMCUs has the same 32-bit affinity value. The 32-bit affinity value is unique within the system as a whole.

The SMIDR_EL1.SH field indicates whether the implementation of Streaming SVE mode is shared with other PEs. However, if SMIDR_EL1.SH is zero, then SMIDR_EL1.Affinity is used to indicate whether the implementation of Streaming SVE is shared, as follows:

- If SMIDR_EL1.Affinity is zero, then the implementation of Streaming SVE mode is not shared with other PEs.
- If SMIDR_EL1.Affinity is not zero, then the implementation of Streaming SVE mode is shared with other PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing SMIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SMIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b001	0b0000	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = SMIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SMIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SMIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SMPRI_EL1, Streaming Mode Priority Register

The SMPRI_EL1 characteristics are:

Purpose

Configures the streaming execution priority for instructions executed on a shared Streaming Mode Compute Unit (SMCU) when the PE is in Streaming SVE mode at any Exception level.

Configuration

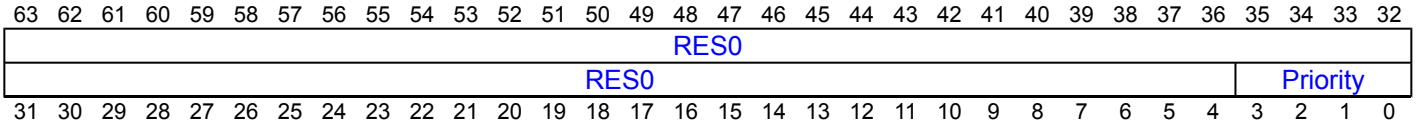
This register is present only when FEAT_SME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SMPRI_EL1 are UNDEFINED.

When SMIDR_EL1.SMPS is '0', this register is RES0.

Attributes

SMPRI_EL1 is a 64-bit register.

Field descriptions



Bits [63:4]

Reserved, RES0.

Priority, bits [3:0]

Streaming execution priority value.

Either this value is used directly, or it is mapped into an effective priority value using SMPRIMAP_EL2.

This value is used directly when any of the following are true:

- The current Exception level is EL3 or EL2.
- The current Exception level is EL1 or EL0, if EL2 is implemented and enabled in the current Security state and HCRX_EL2.SMPME is '0'.
- The current Exception level is EL1 or EL0, if EL2 is either not implemented or not enabled in the current Security state.

The precise meaning and behavior of each streaming execution priority value is IMPLEMENTATION DEFINED.

In an implementation that shares execution resources between PEs, higher priority values are allocated more processing resource than other PEs configured with lower priority values in the same Priority domain.

If FEAT_SME2p2 is implemented, and this field is greater than the Highest Implemented Priority value indicated by SMIDR_EL1.HIP, then the PE uses the Highest Implemented Priority value as the priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMPRI_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SMPRI_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nSMPRI_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SMPRI_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SMPRI_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SMPRI_EL1;
```

```
MSR SMPRI_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nSMPRI_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRI_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRI_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRI_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SMPRIMAP_EL2, Streaming Mode Priority Mapping Register

The SMPRIMAP_EL2 characteristics are:

Purpose

Maps the value in SMPRI_EL1 to a streaming execution priority value for instructions executed at EL1 and EL0 in the same Security states as EL2.

Configuration

This register is present only when FEAT_SME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SMPRIMAP_EL2 are UNDEFINED.

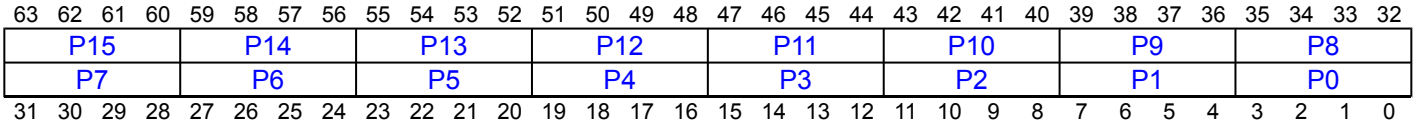
When SMIDR_EL1.SMPS is '0', this register is RES0.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

SMPRIMAP_EL2 is a 64-bit register.

Field descriptions



When all of the following are true, the value in SMPRI_EL1 is mapped to a streaming execution priority using this register:

- The current Exception level is EL1 or EL0.
- EL2 is implemented and enabled in the current Security state.
- HCRX_EL2.SMPME is '1'.

Otherwise, SMPRI_EL1 holds the streaming execution priority value.

P15, bits [63:60]

Priority Mapping Entry 15. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '15'.

This value is the highest streaming execution priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P14, bits [59:56]

Priority Mapping Entry 14. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '14'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P13, bits [55:52]

Priority Mapping Entry 13. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '13'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P12, bits [51:48]

Priority Mapping Entry 12. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '12'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P11, bits [47:44]

Priority Mapping Entry 11. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '11'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P10, bits [43:40]

Priority Mapping Entry 10. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '10'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P9, bits [39:36]

Priority Mapping Entry 9. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '9'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P8, bits [35:32]

Priority Mapping Entry 8. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '8'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P7, bits [31:28]

Priority Mapping Entry 7. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '7'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P6, bits [27:24]

Priority Mapping Entry 6. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '6'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P5, bits [23:20]

Priority Mapping Entry 5. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '5'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P4, bits [19:16]

Priority Mapping Entry 4. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '4'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P3, bits [15:12]

Priority Mapping Entry 3. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '3'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P2, bits [11:8]

Priority Mapping Entry 2. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '2'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P1, bits [7:4]

Priority Mapping Entry 1. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '1'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P0, bits [3:0]

Priority Mapping Entry 0. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '0'.

This value is the lowest streaming execution priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMPRIMAP_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SMPRIMAP_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1F8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SMPRIMAP_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SMPRIMAP_EL2;
```

MSR SMPRIMAP_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1F8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRIMAP_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRIMAP_EL2 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SP_EL0, Stack Pointer (EL0)

The SP_EL0 characteristics are:

Purpose

Holds the stack pointer associated with EL0. At higher Exception levels, this is used as the current stack pointer when the value of SPSel.SP is 0.

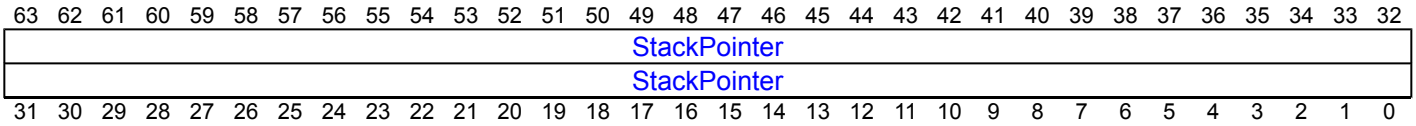
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SP_EL0 are UNDEFINED.

Attributes

SP_EL0 is a 64-bit register.

Field descriptions



StackPointer, bits [63:0]

Stack pointer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SP_EL0

When the value of PSTATE.SP is 0, this register is accessible at all Exception levels as the current stack pointer.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SP_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        X[t, 64] = SP_EL0;
elsif PSTATE.EL == EL2 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        X[t, 64] = SP_EL0;
elsif PSTATE.EL == EL3 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        X[t, 64] = SP_EL0;
```

```
MSR SP_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        SP_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        SP_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        SP_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SP_EL1, Stack Pointer (EL1)

The SP_EL1 characteristics are:

Purpose

Holds the stack pointer associated with EL1. The value of SPSel.SP determines the current stack pointer.

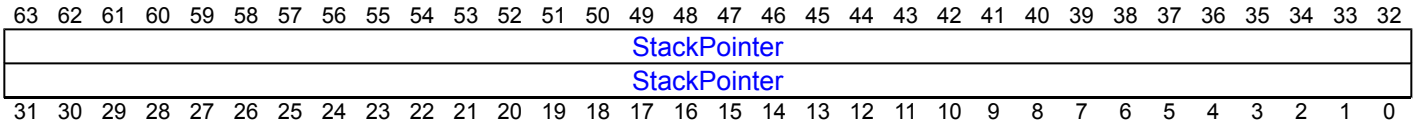
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SP_EL1 are UNDEFINED.

Attributes

SP_EL1 is a 64-bit register.

Field descriptions



StackPointer, bits [63:0]

Stack pointer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SP_EL1

This accessibility information only applies to accesses using the MRS or MSR instructions.

When the value of SPSel.SP is 1, this register is also accessible at EL1 as the current stack pointer.

Note

When the value of SPSel.SP is 0, SP_EL0 is used as the current stack pointer at all Exception levels.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SP_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x240];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SP_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SP_EL1;
```

MSR SP_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x240] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SP_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SP_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SP_EL2, Stack Pointer (EL2)

The SP_EL2 characteristics are:

Purpose

Holds the stack pointer associated with EL2. The value of SPSel.SP determines the current stack pointer.

Configuration

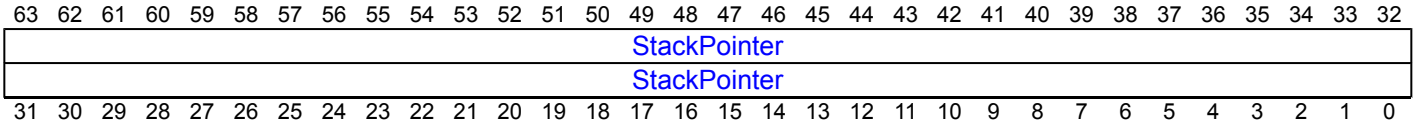
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SP_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

SP_EL2 is a 64-bit register.

Field descriptions



StackPointer, bits [63:0]

Stack pointer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SP_EL2

This accessibility information only applies to accesses using the MRS or MSR instructions.

When the value of SPSel.SP is 1, this register is also accessible at EL2 as the current stack pointer.

Note

When the value of SPSel.SP is 0, SP_EL0 is used as the current stack pointer at all Exception levels.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SP_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0100	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SP_EL2;
```

MSR SP_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b0100	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SP_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SP_EL3, Stack Pointer (EL3)

The SP_EL3 characteristics are:

Purpose

Holds the stack pointer associated with EL3. The value of SPSel.SP determines the current stack pointer.

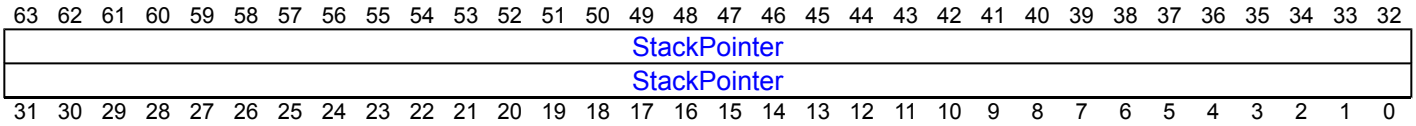
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SP_EL3 are UNDEFINED.

Attributes

SP_EL3 is a 64-bit register.

Field descriptions



StackPointer, bits [63:0]

Stack pointer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SP_EL3

This register is not accessible using MRS and MSR instructions.

When the value of SPSel.SP is 1, this register is accessible at EL3 as the current stack pointer.

Note

When the value of SPSel.SP is 0, SP_EL0 is used as the current stack pointer at all Exception levels.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMACCESSR_EL1, System Performance Monitors Access Register (EL1)

The SPMACCESSR_EL1 characteristics are:

Purpose

Controls access to System PMUs from EL0.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMACCESSR_EL1 are UNDEFINED.

Attributes

SPMACCESSR_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P31		P30		P29		P28		P27		P26		P25		P24		P23		P22		P21		P20		P19		P18		P17		P16	
P15		P14		P13		P12		P11		P10		P9		P8		P7		P6		P5		P4		P3		P2		P1		P0	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

P<m>, bits [2m+1:2m], for m = 31 to 0

System PMU <m> access. Controls access to System PMU <m>.

P<m>	Meaning
0b00	MRS read and MSR write System register accesses to System PMU <m> at EL0 are trapped to EL1, unless the instruction generates a higher priority exception.
0b01	MSR write System register accesses to System PMU <m> at EL0 are trapped to EL1, unless the instruction generates a higher priority exception.
0b11	This control does not cause any instructions to be trapped.

All other values are reserved.

The registers trapped by this control are:

AArch64: SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMOVSCLR_EL0, and SPMOVSSET_EL0.

This field is ignored by the PE when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H,TGE} is {1,1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m > UInt(ID_AA64DFR1_EL1.SYSPMUID), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing SPMACCESSR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name SPMACCESSR_EL1 or SPMACCESSR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMACCESSR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMACCESSR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8E8];
    else
        X[t, 64] = SPMACCESSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SPMACCESSR_EL2;
    else
        X[t, 64] = SPMACCESSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMACCESSR_EL1;
```

```
MSR SPMACCESSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMACCESSR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8E8] = X[t, 64];
    else
        SPMACCESSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        SPMACCESSR_EL2 = X[t, 64];
    else
        SPMACCESSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMACCESSR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, SPMACCESSR_EL12
```

op0	op1	CRn	CRm	op2
0b10	0b101	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x8E8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SPMACCESSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SPMACCESSR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR SPMACCESSR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b101	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x8E8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SPMACCESSR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SPMACCESSR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMACCESSR_EL2, System Performance Monitors Access Register (EL2)

The SPMACCESSR_EL2 characteristics are:

Purpose

Controls access to System PMUs from EL1 and EL0.

Configuration

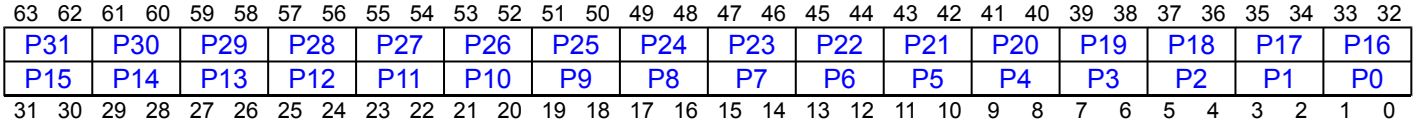
This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMACCESSR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

SPMACCESSR_EL2 is a 64-bit register.

Field descriptions



P<m>, bits [2m+1:2m], for m = 31 to 0

System PMU <m> access. Controls access to System PMU <m>.

P<m>	Meaning
0b00	MRS read and MSR write System register accesses to System PMU <m> at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.
0b01	MSR write System register accesses to System PMU <m> at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.
0b11	This control does not cause any instructions to be trapped.

All other values are reserved.

The registers trapped by this control are:

AArch64: SPMCFGR_EL1, SPMCGCR<n>_EL1, SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMDEVAFF_EL1, SPMDEVARCH_EL1, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMIIDR_EL1, SPMINTENCLR_EL1, SPMINTENSET_EL1, SPMOVSCLR_EL0, SPMOVSSET_EL0, and SPMSCR_EL1.

This field is ignored by the PE when EL2 is not implemented or disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m > UInt(ID_AA64DFR1_EL1.SYSPMUID), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing SPMACCESSR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SPMACCESSR_EL2 or SPMACCESSR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMACCESSR_EL2
```

op0	op1	CRn	CRm	op2
0b10	0b100	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMACCESSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMACCESSR_EL2;
```

```
MSR SPMACCESSR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b100	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMACCESSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMACCESSR_EL2 = X[t, 64];
```

```
MRS <Xt>, SPMACCESSR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMACCESSR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8E8];
    else
        X[t, 64] = SPMACCESSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SPMACCESSR_EL2;
    else
        X[t, 64] = SPMACCESSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMACCESSR_EL1;
```

```
MSR SPMACCESSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMACCESSR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8E8] = X[t, 64];
    else
        SPMACCESSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        SPMACCESSR_EL2 = X[t, 64];
    else
        SPMACCESSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMACCESSR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMACCESSR_EL3, System Performance Monitors Access Register (EL3)

The SPMACCESSR_EL3 characteristics are:

Purpose

Controls access to System PMUs from EL2, EL1 and EL0.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMACCESSR_EL3 are UNDEFINED.

Attributes

SPMACCESSR_EL3 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P31		P30		P29		P28		P27		P26		P25		P24		P23		P22		P21		P20		P19		P18		P17		P16	
P15		P14		P13		P12		P11		P10		P9		P8		P7		P6		P5		P4		P3		P2		P1		P0	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

P<m>, bits [2m+1:2m], for m = 31 to 0

System PMU <m> access. Controls access to System PMU <m>.

P<m>	Meaning
0b00	MRS read and MSR write System register accesses to System PMU <m> at EL2, EL1, and EL0 are trapped to EL3, unless the instruction generates a higher priority exception.
0b01	MSR write System register accesses to System PMU <m> at EL2, EL1, and EL0 are trapped to EL3, unless the instruction generates a higher priority exception.
0b11	This control does not cause any instructions to be trapped.

All other values are reserved.

The registers trapped by this control are:

AArch64: SPMCFGR_EL1, SPMCGCR<n>_EL1, SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMDEVAFF_EL1, SPMDEVARCH_EL1, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMIIDR_EL1, SPMINTENCLR_EL1, SPMINTENSET_EL1, SPMOVSCLR_EL0, SPMOVSSET_EL0, and SPMSCR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m > UInt(ID_AA64DFR1_EL1.SYSPMUID), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing SPMACCESSR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SPMACCESSR_EL3

op0	op1	CRn	CRm	op2
0b10	0b110	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMACCESSR_EL3;
```

MSR SPMACCESSR_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b110	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SPMACCESSR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMCFGR_EL1, System Performance Monitors Configuration Register

The SPMCFGR_EL1 characteristics are:

Purpose

Describes the capabilities of System PMU <s>.

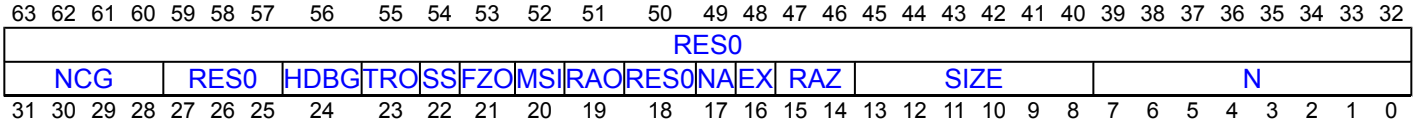
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMCFGR_EL1 are UNDEFINED.

Attributes

SPMCFGR_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NCG, bits [31:28]

Counter Groups.

Defines the number of counter groups implemented by System PMU <s>, minus one.

If this field is zero, then one counter group is implemented and SPMCGCR<n>_EL1 read-as-zero.

Otherwise, for each counter group <m>, SPMCGCR<m DIV 8>_EL1.N<m MOD 8> defines the number of counters in the group.

Locating the first counter in each group depends on the number of implemented groups. Each counter group starts with counter:

- SPMEVTYPER<m×32>_EL0, meaning there are at most 32 counters per group, if there are 2 counter groups.
- SPMEVTYPER<m×16>_EL0, meaning there are at most 16 counters per group, if there are 3 or 4 counter groups.
- SPMEVTYPER<m×8>_EL0, meaning there are at most 8 counters per group, if there are between 5 and 8 counter groups.
- SPMEVTYPER<m×4>_EL0, meaning there are at most 4 counters per group, if there are more than 8 counter groups.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [27:25]

Reserved, RES0.

HDBG, bit [24]

Halt-on-debug supported. For more information on this field, see 'CoreSight PMU Architecture'.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

TRO, bit [23]

Trace output supported. For more information on this field, see 'CoreSight PMU Architecture'.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SS, bit [22]

Snapshot supported. For more information on this field, see 'CoreSight PMU Architecture'.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

FZO, bit [21]

Freeze-on-overflow supported. For more information on this field, see 'CoreSight PMU Architecture'.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

MSI, bit [20]

Message-signaled interrupts supported. For more information on this field, see 'CoreSight PMU Architecture'.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [19]

Reserved, RAO.

Bit [18]

Reserved, RES0.

NA, bit [17]

No write access when running. For more information on this field, see 'CoreSight PMU Architecture'.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

EX, bit [16]

Export supported. For more information on this field, see 'CoreSight PMU Architecture'.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [15:14]

Reserved, RAZ.

SIZE, bits [13:8]

Counter size. The size of the largest counter implemented by System PMU <s>.

SIZE	Meaning
0b000111	8-bit counters.
0b001001	10-bit counters.
0b001011	12-bit counters.
0b001111	16-bit counters.
0b010011	20-bit counters.
0b010111	24-bit counters.
0b011111	32-bit counters.
0b100011	36-bit counters.
0b100111	40-bit counters.
0b101011	44-bit counters.
0b101111	48-bit counters.
0b110011	52-bit counters.
0b110111	56-bit counters.
0b111111	64-bit counters.

All other values are reserved.

Not all counters must be this size. For example, a System PMU might include a mix of 32-bit and 64-bit counters.

N, bits [7:0]

Number of event counters implemented by System PMU <s>, minus 1.

N	Meaning
0x00..0x3F	Number of event counters implemented by System PMU <s>, minus 1.

All other values are reserved.

Accessing SPMCFGR_EL1

To access SPMCFGR_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMCFGR_EL1 reads-as-zero if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMCFGR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b111

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMID == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCFGR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCFGR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMCFGR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMCGCR<n>_EL1, System PMU Counter Group Configuration Registers, n = 0 - 1

The SPMCGCR<n>_EL1 characteristics are:

Purpose

Describes the configuration of counter groups in System PMU <s>.

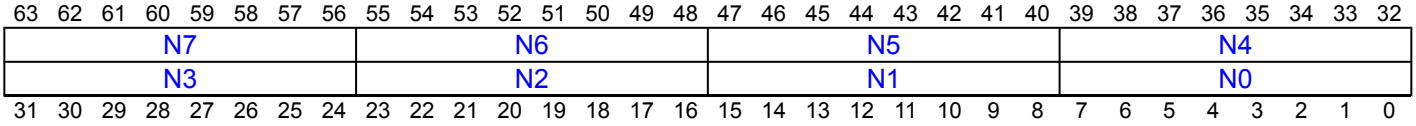
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMCGCR<n>_EL1 are UNDEFINED.

Attributes

SPMCGCR<n>_EL1 is a 64-bit register.

Field descriptions



N<m>, bits [8m+7:8m], for m = 7 to 0

Number of counters in counter group 8n+m.

The maximum size of each counter group depends on the number of implemented groups and the largest implemented counter size. For more information, see SPMCFGR_EL1.NCG.

Accessing SPMCGCR<n>_EL1

To access SPMCGCR<n>_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMCGCR<n>_EL1 reads-as-zero if any of the following are true:

- System PMU <s> implements one counter group (SPMCFGR_EL1.NCG is zero).
- System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMCGCR<m>_EL1 ; Where m = 0-1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b00:m[0]

```
integer m = UInt(op2<0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMID == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCGCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL), m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCGCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL), m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMCGCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL), m];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMCNTENCLR_EL0, System Performance Monitors Count Enable Clear Register

The SPMCNTENCLR_EL0 characteristics are:

Purpose

Disable event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMCNTENCLR_EL0 are UNDEFINED.

Attributes

SPMCNTENCLR_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

P<m>, bit [m], for m = 63 to 0

Event counter <m> disable.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> is disabled.
0b1	Event counter <m> in System PMU <s> is enabled.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When event counter <m> is not implemented by System PMU <s>, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing SPMCNTENCLR_EL0

To access SPMCNTENCLR_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMCNTENCLR_EL0 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMCNTENCLR_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMCNTENCLR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMCNTENSET_EL0, System Performance Monitors Count Enable Set Register

The SPMCNTENSET_EL0 characteristics are:

Purpose

Enables event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMCNTENSET_EL0 are UNDEFINED.

Attributes

SPMCNTENSET_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

P<m>, bit [m], for m = 63 to 0

Event counter <m> enable.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> is disabled.
0b1	Event counter <m> in System PMU <s> is enabled.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When event counter <m> is not implemented by System PMU <s>, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing SPMCNTENSET_EL0

To access SPMCNTENSET_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMCNTENSET_EL0 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMCNTENSET_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMCNTENSET_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMCR_EL0, System Performance Monitor Control Register

The SPMCR_EL0 characteristics are:

Purpose

Main control register for System PMU <s>.

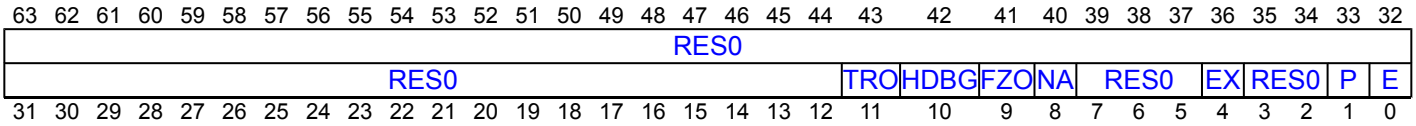
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMCR_EL0 are UNDEFINED.

Attributes

SPMCR_EL0 is a 64-bit register.

Field descriptions



Bits [63:12]

Reserved, RES0.

TRO, bit [11]

When SPMCFGR_EL1.TRO == 1:

Trace enable. For more information on this field, see 'CoreSight PMU Architecture'.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When PSTATE.EL == EL0, access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

HDBG, bit [10]

When SPMCFGR_EL1.HDBG == 1:

Halt-on-debug. For more information on this field, see 'CoreSight PMU Architecture'.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When PSTATE.EL == EL0, access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

FZO, bit [9]
When SPMCFGR_EL1.FZO == 1:

Freeze-on-overflow. For more information on this field, see 'CoreSight PMU Architecture'.

Note

If implemented by a System PMU, then freeze-on-overflow affects only the counters of System PMU <s>, not other System PMUs nor the PE PMU.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NA, bit [8]
When SPMCFGR_EL1.NA == 1:

Not accessible. For more information on this field, see 'CoreSight PMU Architecture'.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [7:5]

Reserved, RES0.

EX, bit [4]
When SPMCFGR_EL1.EX == 1:

Export enable. For more information on this field, see 'CoreSight PMU Architecture'.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [3:2]

Reserved, RES0.

P, bit [1]

Event counter reset.

P	Meaning
0b0	Write is ignored.
0b1	Reset all event counters in System PMU <s> to zero.

Note

Resetting the event counters does not affect any overflow flags.

Access to this field is **WO/RAZ**.

E, bit [0]

Count enable. This field controls System PMU <s>.

E	Meaning
0b0	Monitor is disabled.
0b1	Monitor is enabled.

Performance monitor overflow IRQs are only signaled by System PMU <s> when this field is 1.

The reset behavior of this field is:

- On a System PMU reset, this field resets to '0'.

Accessing SPMCR_EL0

To access SPMCR_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMCR_EL0 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMCR_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMCR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMCR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMCR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMCR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMCR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMDEVAFF_EL1, System Performance Monitors Device Affinity Register

The SPMDEVAFF_EL1 characteristics are:

Purpose

For additional information, see the CoreSight Architecture Specification.

For a System PMU that has affinity with a single PE or a group of PEs, SPMDEVAFF_EL1 is a copy of MPIDR_EL1 or part of MPIDR_EL1:

- If the System PMU has affinity with a single PE, then the affinity level is 0 and SPMDEVAFF_EL1 reads the same value as MPIDR_EL1, and SPMDEVAFF_EL1.F0V reads-as-one to indicate affinity level 0.
- If the System PMU has affinity with a group of PEs, then the affinity level is 1, 2, or 3, parts of SPMDEVAFF_EL1 reads the same value as parts of MPIDR_EL1, and the rest of SPMDEVAFF_EL1 indicates the level.

For example, if the group of PEs is a subset of the PEs at affinity level 1 then all of the following are true:

- All the PEs in the group have the same values in MPIDR_EL1.{Aff3,Aff2}, and these values are equal to SPMDEVAFF_EL1.{Aff3,Aff2}.
- SPMDEVAFF_EL1.Aff1 is nonzero and not `0x80`, and SPMDEVAFF_EL1.{Aff0,F0V} read-as-zero, to indicate at least affinity level 1. The subset of PEs at level 1 that the System PMU has affinity with is indicated by the least-significant set bit in SPMDEVAFF_EL1.Aff1. In this example, if SPMDEVAFF_EL1.Aff1[2:0] is `0b100`, then the System PMU has affinity with the up-to 8 PEs that have MPIDR_EL1.Aff1[7:3] == SPMDEVAFF_EL1.Aff1[7:3].

Depending on the IMPLEMENTATION DEFINED nature of the system, it might be possible that SPMDEVAFF_EL1 is read before system firmware has configured the System PMU and/or the PE or group of PEs that the System PMU has affinity with. When this is the case, SPMDEVAFF_EL1 reads-as-zero.

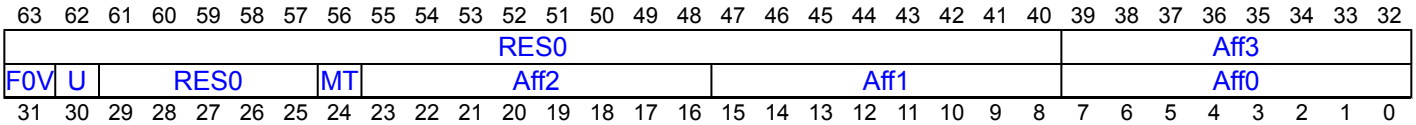
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMDEVAFF_EL1 are UNDEFINED.

Attributes

SPMDEVAFF_EL1 is a 64-bit register.

Field descriptions



Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

PE affinity level 3. The MPIDR_EL1.Aff3 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

F0V, bit [31]

Indicates that the SPMDEVAFF_EL1.Aff0 field is valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F0V	Meaning
0b0	SPMDEVAFF_EL1.Aff0 is not valid, and the PE affinity is above level 0 or a subset of level 0.
0b1	SPMDEVAFF_EL1.Aff0 is valid, and the PE affinity is at level 0.

Access to this field is **RO**.

U, bit [30]

When SPMDEVAFF_EL1.F0V == 1:

Uniprocessor. The MPIDR_EL1.U field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, UNKNOWN.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

When SPMDEVAFF_EL1.F0V == 1:

Multithreaded. The MPIDR_EL1.MT field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, UNKNOWN.

Aff2, bits [23:16]

When affine with a PE or PEs at affinity level 2 or below:

PE affinity level 2. The MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

When affine with a sub-set of PEs at affinity level 2:

PE affinity level 2. Defines part of the MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff2	Meaning
0bxxxxxxx1	SPMDEVAFF_EL1.Aff2[7:1] is the value of MPIDR_EL1.Aff2[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	SPMDEVAFF_EL1.Aff2[7:2] is the value of MPIDR_EL1.Aff2[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	SPMDEVAFF_EL1.Aff2[7:3] is the value of MPIDR_EL1.Aff2[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	SPMDEVAFF_EL1.Aff2[7:4] is the value of MPIDR_EL1.Aff2[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	SPMDEVAFF_EL1.Aff2[7:5] is the value of MPIDR_EL1.Aff2[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	SPMDEVAFF_EL1.Aff2[7:6] is the value of MPIDR_EL1.Aff2[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	SPMDEVAFF_EL1.Aff2[7] is the value of MPIDR_EL1.Aff2[7], viewed from the highest Exception level of the associated PEs.

Access to this field is **RO**.

Otherwise:

PE affinity level NOT DEFINED. Indicates whether the PE affinity is at level 3.

Aff2	Meaning
0x80	PE affinity is at level 3.

All other values are reserved.

Access to this field is **RO**.

Aff1, bits [15:8]

When affine with a PE or PEs at affinity level 1 or below:

PE affinity level 1. The MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

When affine with a sub-set of PEs at affinity level 1:

PE affinity level 1. Defines part of the MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff1	Meaning
0bxxxxxxx1	SPMDEVAFF_EL1.Aff1[7:1] is the value of MPIDR_EL1.Aff1[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	SPMDEVAFF_EL1.Aff1[7:2] is the value of MPIDR_EL1.Aff1[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	SPMDEVAFF_EL1.Aff1[7:3] is the value of MPIDR_EL1.Aff1[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	SPMDEVAFF_EL1.Aff1[7:4] is the value of MPIDR_EL1.Aff1[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	SPMDEVAFF_EL1.Aff1[7:5] is the value of MPIDR_EL1.Aff1[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	SPMDEVAFF_EL1.Aff1[7:6] is the value of MPIDR_EL1.Aff1[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	SPMDEVAFF_EL1.Aff1[7] is the value of MPIDR_EL1.Aff1[7], viewed from the highest Exception level of the associated PEs.

Access to this field is **RO**.

Otherwise:

PE affinity level 1. Indicates whether the PE affinity is at level 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff1	Meaning
0x00	PE affinity is above level 2 or a subset of level 2.
0x80	PE affinity is at level 2.

Access to this field is **RO**.

Aff0, bits [7:0]

When affine with a PE at affinity level 0:

PE affinity level 0. The MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

When affine with a sub-set of PEs at affinity level 0:

PE affinity level 0. Defines part of the MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff0	Meaning
0bxxxxxxx1	SPMDEVAFF_EL1.Aff0[7:1] is the value of MPIDR_EL1.Aff0[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	SPMDEVAFF_EL1.Aff0[7:2] is the value of MPIDR_EL1.Aff0[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	SPMDEVAFF_EL1.Aff0[7:3] is the value of MPIDR_EL1.Aff0[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	SPMDEVAFF_EL1.Aff0[7:4] is the value of MPIDR_EL1.Aff0[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	SPMDEVAFF_EL1.Aff0[7:5] is the value of MPIDR_EL1.Aff0[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	SPMDEVAFF_EL1.Aff0[7:6] is the value of MPIDR_EL1.Aff0[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	SPMDEVAFF_EL1.Aff0[7] is the value of MPIDR_EL1.Aff0[7], viewed from the highest Exception level of the associated PEs.

Access to this field is **RO**.

Otherwise:

PE affinity level 0. Indicates whether the PE affinity is at level 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff0	Meaning
0x00	PE affinity is above level 1 or a subset of level 1.
0x80	PE affinity is at level 1.

Access to this field is **RO**.

Accessing SPMDEVAFF_EL1

Reads of SPMDEVAFF_EL1 are not affected by the value of VMPIDR_EL2 at any Exception level.

If System PMU <s> has affinity only with this PE, then it is IMPLEMENTATION DEFINED whether SPMDEVAFF_EL1 reads-as-zero or reads the same value as MPIDR_EL1.

To access SPMDEVAFF_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMDEVAFF_EL1 reads-as-zero if any of the following are true:

- System PMU <s> is not implemented.
- System PMU <s> has no affinity with the PE or cluster of PEs.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMDEVAFF_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b110

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMDEVAFF_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMDEVAFF_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMDEVAFF_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMDEVAFF_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMDEVARCH_EL1, System Performance Monitors Device Architecture Register

The SPMDEVARCH_EL1 characteristics are:

Purpose

Provides discovery information for System PMU <s>.

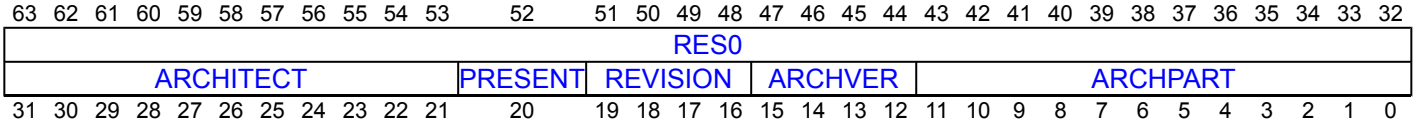
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMDEVARCH_EL1 are UNDEFINED.

Attributes

SPMDEVARCH_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

ARCHITECT, bits [31:21]

Architect. Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PRESENT, bit [20]

DEVARCH present. Defines that SPMDEVARCH_EL1 register is present.

PRESENT	Meaning
0b0	Device Architecture information not present.
0b1	Device Architecture information present.

If SPMDEVARCH_EL1 is not present, the register is RES0.

REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

SPMDEVARCH_EL1.ARCHVER and SPMDEVARCH_EL1.ARCHPART are also defined as a single field, SPMDEVARCH_EL1.ARCHID, so that SPMDEVARCH_EL1.ARCHVER is SPMDEVARCH_EL1.ARCHID[15:12].

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

SPMDEVARCH_EL1.ARCHVER and SPMDEVARCH_EL1.ARCHPART are also defined as a single field, SPMDEVARCH_EL1.ARCHID, so that SPMDEVARCH_EL1.ARCHPART is SPMDEVARCH_EL1.ARCHID[11:0].

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing SPMDEVARCH_EL1

To access SPMDEVARCH_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMDEVARCH_EL1 reads-as-zero if any of the following are true:

- System PMU <s> is not implemented.
- System PMU <s> does not implement SPMDEVARCH_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMDEVARCH_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b101

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMID == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMDEVARCH_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMDEVARCH_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMDEVARCH_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMEVCNTR<n>_EL0, System Performance Monitors Event Count Register, n = 0 - 63

The SPMEVCNTR<n>_EL0 characteristics are:

Purpose

Event counter <n> in System PMU <s>, where n is 0 to 63.

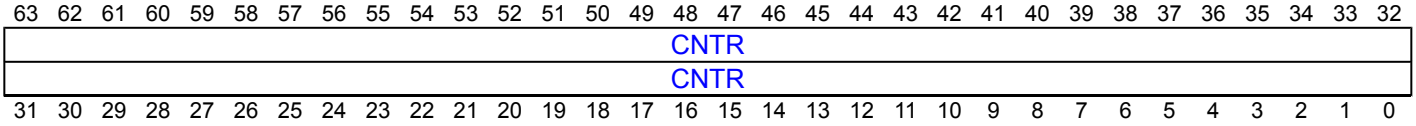
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMEVCNTR<n>_EL0 are UNDEFINED.

Attributes

SPMEVCNTR<n>_EL0 is a 64-bit register.

Field descriptions



CNTR, bits [63:0]

Event counter n.

The number of implemented bits for SPMEVCNTR<n>_EL0 is IMPLEMENTATION DEFINED. Unimplemented bits are RES0.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing SPMEVCNTR<n>_EL0

To access SPMEVCNTR<n>_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s and SPMSELR_EL0.BANK to n[5:4].

SPMEVCNTR<n>_EL0 reads-as-zero and ignores writes if any of the following are true:

- Event counter <n> is not implemented by System PMU <s>.
- System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMEVCNTR<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b000:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m)
then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
MSR SPMEVCNTR<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b000:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m)
then
        return;
    else
        SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMEVFILT2R<n>_EL0, System Performance Monitors Event Filter Control Register 2, n = 0 - 63

The SPMEVFILT2R<n>_EL0 characteristics are:

Purpose

With SPMEVTYPER<n>_EL0 and SPMEVFILTR<n>_EL0, configures when event counter SPMEVCNTR<n>_EL0 in System PMU <s> increments.

The contents of this register are IMPLEMENTATION DEFINED. For more information, see SPMEVTYPER<n>_EL0.

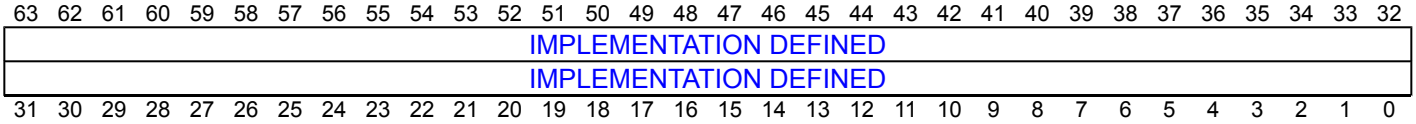
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMEVFILT2R<n>_EL0 are UNDEFINED.

Attributes

SPMEVFILT2R<n>_EL0 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing SPMEVFILT2R<n>_EL0

To access SPMEVFILT2R<n>_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s and SPMSELR_EL0.BANK to n[5:4].

SPMEVFILT2R<n>_EL0 reads-as-zero and ignores writes if any of the following are true:

- Event counter <n> is not implemented by System PMU <s>.
- System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMEVFILT2R<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b011:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m)
then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
MSR SPMEVFILT2R<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b011:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m)
then
        return;
    else
        SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMEVFILTR<n>_EL0, System Performance Monitors Event Filter Control Register, n = 0 - 63

The SPMEVFILTR<n>_EL0 characteristics are:

Purpose

With SPMEVTYPER<n>_EL0 and SPMEVFILT2R<n>_EL0, configures when event counter SPMEVCNTR<n>_EL0 in System PMU <s> increments.

The contents of this register are IMPLEMENTATION DEFINED. For more information, see SPMEVTYPER<n>_EL0.

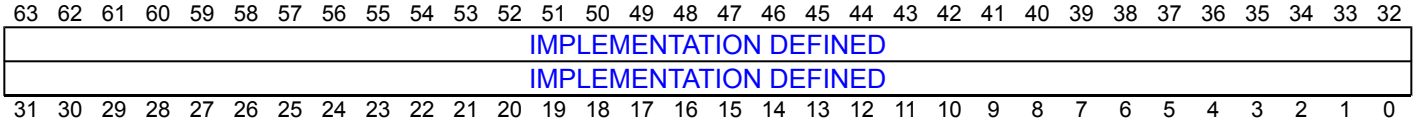
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMEVFILTR<n>_EL0 are UNDEFINED.

Attributes

SPMEVFILTR<n>_EL0 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing SPMEVFILTR<n>_EL0

To access SPMEVFILTR<n>_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s and SPMSELR_EL0.BANK to n[5:4].

SPMEVFILTR<n>_EL0 reads-as-zero and ignores writes if any of the following are true:

- Event counter <n> is not implemented by System PMU <s>.
- System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMEVFILTR<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b010:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m)
then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
MSR SPMEVFILTR<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b010:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m)
then
        return;
    else
        SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMEVTYPER<n>_EL0, System Performance Monitors Event Type Register, n = 0 - 63

The SPMEVTYPER<n>_EL0 characteristics are:

Purpose

With SPMEVFILTR<n>_EL0 and SPMEVFILT2R<n>_EL0, configures when event counter SPMEVCNTR<n>_EL0 in System PMU <s> increments.

The contents of this register are IMPLEMENTATION DEFINED. An Event Type Select Register typically contains:

- A field defining the event that the counter is responsive to, in the least-significant bits.
- Controls for per-counter filtering, such as by mode or state.
- Additional controls, such as for a per-counter state machine.

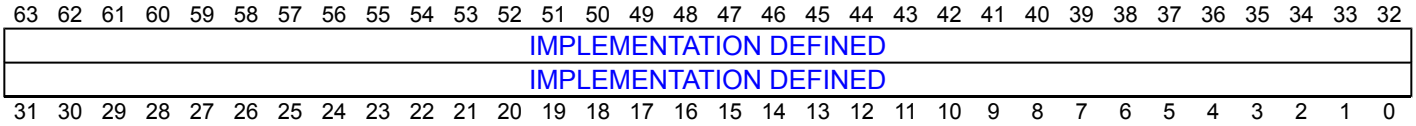
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMEVTYPER<n>_EL0 are UNDEFINED.

Attributes

SPMEVTYPER<n>_EL0 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing SPMEVTYPER<n>_EL0

To access SPMEVTYPER<n>_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s and SPMSELR_EL0.BANK to n[5:4].

SPMEVTYPER<n>_EL0 reads-as-zero and ignores writes if any of the following are true:

- Event counter <n> is not implemented by System PMU <s>.
- System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMEVTYPER<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b001:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m)
then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
MSR SPMEVTYPER<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b001:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) +
m) then
        return;
    else
        SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m)
then
        return;
    else
        SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMIIDR_EL1, System PMU Implementation Identification Register

The SPMIIDR_EL1 characteristics are:

Purpose

Provides discovery information for System PMU <s>.

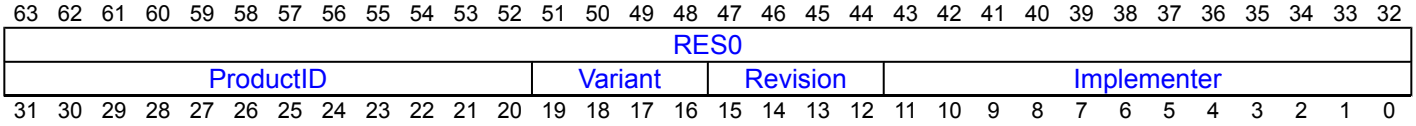
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMIIDR_EL1 are UNDEFINED.

Attributes

SPMIIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

ProductID, bits [31:20]

Part number, bits [11:0]. The part number is selected by the designer of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [19:16]

Component major revision.

Defines either a variant of the component defined by SPMIIDR_EL1.ProductID, or the major revision of the component.

When defining a major revision, SPMIIDR_EL1.Variant and SPMIIDR_EL1.Revision together form the revision number of the component, with SPMIIDR_EL1.Variant being the most significant part and SPMIIDR_EL1.Revision the least significant part. When a component is changed, SPMIIDR_EL1.Variant or SPMIIDR_EL1.Revision is increased to ensure that software can differentiate the different revisions of the component. If SPMIIDR_EL1.Variant is increased then SPMIIDR_EL1.Revision should be set to `0b0000`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Component minor revision.

When a component is changed:

- If SPMIIDR_EL1.Variant and SPMIIDR_EL1.Revision together form the revision number of the component then:
  - SPMIIDR_EL1.Variant or SPMIIDR_EL1.Revision is increased to ensure that software can differentiate the different revisions of the component.
  - If Variant is increased then Revision should be set to `0b0000`.
- Otherwise, SPMIIDR_EL1.Revision is increased to ensure that software can differentiate the different revisions of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the System PMU.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing SPMIIDR_EL1

To access SPMIIDR_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMIIDR_EL1 reads-as-zero if any of the following are true:

- System PMU <s> is not implemented.
- System PMU <s> does not implement SPMIIDR_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMIIDR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b100

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMID == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMIIDR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMIIDR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMIIDR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMINTENCLR_EL1, System Performance Monitors Interrupt Enable Clear Register

The SPMINTENCLR_EL1 characteristics are:

Purpose

Disables the generation of interrupt requests on overflows from event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMINTENCLR_EL1 are UNDEFINED.

Attributes

SPMINTENCLR_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

P<m>, bit [m], for m = 63 to 0

Event counter <m> overflow interrupt request disable.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> interrupt request is disabled.
0b1	Event counter <m> in System PMU <s> interrupt request is enabled.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if any the following are true:
  - event counter <m> is not implemented by System PMU <s>.
  - event counter <m> does not implement an overflow flag.
  - System PMU <s> does not implement an overflow interrupt request.
- Otherwise, access to this field is **W1C**.

Accessing SPMINTENCLR_EL1

To access SPMINTENCLR_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMINTENCLR_EL1 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMINTENCLR_EL1
```

op0	op1	CRn	CRm	op2

0b10	0b000	0b1001	0b1110	0b010

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMINTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMINTENCLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1110	0b010

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMINTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMINTENSET_EL1, System Performance Monitors Interrupt Enable Set Register

The SPMINTENSET_EL1 characteristics are:

Purpose

Enables the generation of interrupt requests on overflows from event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMINTENSET_EL1 are UNDEFINED.

Attributes

SPMINTENSET_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

P<m>, bit [m], for m = 63 to 0

Event counter <m> overflow interrupt request enable.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> interrupt request is disabled.
0b1	Event counter <m> in System PMU <s> interrupt request is enabled.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if any the following are true:
  - event counter <m> is not implemented by System PMU <s>.
  - event counter <m> does not implement an overflow flag.
  - System PMU <s> does not implement an overflow interrupt request.
- Otherwise, access to this field is **W1S**.

Accessing SPMINTENSET_EL1

To access SPMINTENSET_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMINTENSET_EL1 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMINTENSET_EL1
```

op0	op1	CRn	CRm	op2

0b10	0b000	0b1001	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMINTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMINTENSET_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMINTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMOVSCLR_EL0, System Performance Monitors Overflow Flag Status Clear Register

The SPMOVSCLR_EL0 characteristics are:

Purpose

Clears the state of overflow bits for event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMOVSCLR_EL0 are UNDEFINED.

Attributes

SPMOVSCLR_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

P<m>, bit [m], for m = 63 to 0

Event counter <m> unsigned overflow bit clear.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> has not overflowed.
0b1	Event counter <m> in System PMU <s> has overflowed.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if any the following are true:
  - event counter <m> is not implemented by System PMU <s>.
  - event counter <m> does not implement an overflow flag.
- Otherwise, access to this field is **W1C**.

Accessing SPMOVSCLR_EL0

To access SPMOVSCLR_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMOVSCLR_EL0 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMOVSCLR_EL0
```

op0	op1	CRn	CRm	op2

0b10	0b011	0b1001	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMOVSCLR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMOVSSET_EL0, System Performance Monitors Overflow Flag Status Set Register

The SPMOVSSET_EL0 characteristics are:

Purpose

Sets the state of overflow bits for event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMOVSSET_EL0 are UNDEFINED.

Attributes

SPMOVSSET_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

P<m>, bit [m], for m = 63 to 0

Event counter <m> unsigned overflow bit set.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> has not overflowed.
0b1	Event counter <m> in System PMU <s> has overflowed.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if any the following are true:
  - event counter <m> is not implemented by System PMU <s>.
  - event counter <m> does not implement an overflow flag.
- Otherwise, access to this field is **W1S**.

Accessing SPMOVSSET_EL0

To access SPMOVSSET_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMOVSSET_EL0 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMOVSSET_EL0
```

op0	op1	CRn	CRm	op2

0b10	0b011	0b1001	0b1110	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMOVSSET_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1110	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMROOTCR_EL3, System Performance Monitors Root and Realm Control Register

The SPMROOTCR_EL3 characteristics are:

Purpose

Controls observability of Root and Realm events by System PMU <s>.

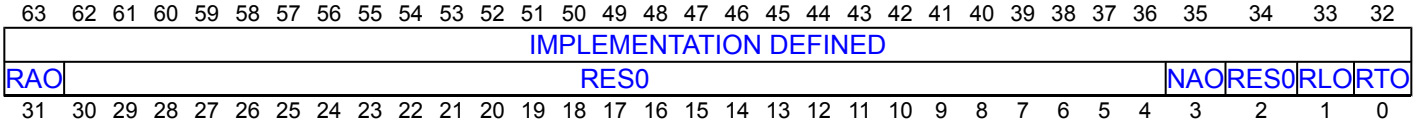
Configuration

This register is present only when FEAT_RME is implemented, FEAT_SPMU is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMROOTCR_EL3 are UNDEFINED.

Attributes

SPMROOTCR_EL3 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:32]

IMPLEMENTATION DEFINED observation controls. Additional IMPLEMENTATION DEFINED bits to control certain types of filter or events by System PMU <s>.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an IMPLEMENTATION DEFINED value.

Bit [31]

Reserved, RAO.

Indicates SPMROOTCR_EL3 is implemented by System PMU <s>.

Bits [30:4]

Reserved, RES0.

NAO, bit [3]

When System PMU <s> can count or monitor non-attributable events:

Non-attributable Observation. Controls whether events or monitorable characteristics not attributable with any source can be monitored by System PMU <s>.

NAO	Meaning
0b0	Events not attributable with any event source are not counted by System PMU <s>.
0b1	Counting non-attributable events by System PMU <s> is not prevented by this mechanism.

When both SPMROOTCR_EL3 and SPMSCR_EL1 are implemented, non-attributable events are counted only if both SPMROOTCR_EL3.NAO is 1 and SPMSCR_EL1.{NAO, SO} is nonzero.

SPMROOTCR_EL3.NAO has the opposite reset polarity to SPMSCR_EL1.NAO.

The reset behavior of this field is:

- On a System PMU reset, this field resets to `'1'`.

Otherwise:

Reserved, RES0.

Bit [2]

Reserved, RES0.

RLO, bit [1]

Realm Observation. Controls whether events or monitorable characteristics attributable to a Realm event source can be monitored by System PMU <s>.

RLO	Meaning
`0b0`	Events attributable to a Realm event source are not counted by System PMU <s>.
`0b1`	Counting events by System PMU <s> that are attributable to a Realm event source is not prevented by this mechanism.

The reset behavior of this field is:

- On a System PMU reset, this field resets to `'0'`.

RTO, bit [0]

Root Observation. Controls whether events or monitorable characteristics attributable to a Root event source can be monitored by System PMU <s>.

RTO	Meaning
`0b0`	Events attributable to a Root event source are not counted by System PMU <s>.
`0b1`	Counting events by System PMU <s> that are attributable to a Root event source is not prevented by this mechanism.

The reset behavior of this field is:

- On a System PMU reset, this field resets to `'0'`.

Accessing SPMROOTCR_EL3

To access SPMROOTCR_EL3 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMROOTCR_EL3 reads-as-zero and ignores writes if any of the following are true:

- System PMU <s> is not implemented.
- System PMU <s> does not implement SPMROOTCR_EL3.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMROOTCR_EL3
```

op0	op1	CRn	CRm	op2
0b10	0b110	0b1001	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_SPMU) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMROOTCR_EL3[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMROOTCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b110	0b1001	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_SPMU) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.SPMROOTCR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMROOTCR_EL3[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMSCR_EL1, System Performance Monitors Secure Control Register

The SPMSCR_EL1 characteristics are:

Purpose

Controls observability of Secure events by System PMU <s>, and optionally controls Secure attributes for message signaled interrupts and Non-secure access to the performance monitor registers.

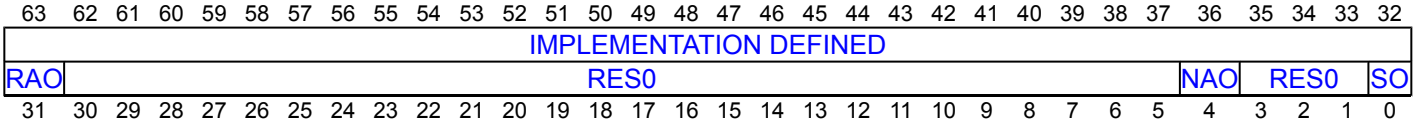
Configuration

This register is present only when Secure EL1 is implemented, FEAT_SPMU is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMSCR_EL1 are UNDEFINED.

Attributes

SPMSCR_EL1 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:32]

IMPLEMENTATION DEFINED observation controls. Additional IMPLEMENTATION DEFINED bits to control certain types of filter or events by System PMU <s>.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an IMPLEMENTATION DEFINED value.

Bit [31]

Reserved, RAO.

Indicates SPMSCR_EL1 is implemented by System PMU <s>.

This field reads-as-one.

Bits [30:5]

Reserved, RES0.

NAO, bit [4]

When System PMU <s> can count or monitor non-attributable events:

Non-attributable Observation. Controls whether events or monitorable characteristics not attributable with any source can be monitored by System PMU <s>.

NAO	Meaning
0b0	Events not attributable with any event source are not counted by System PMU <s>, unless overridden by SPMSCR_EL1.SO.
0b1	Counting non-attributable events by System PMU <s> is not prevented by this mechanism.

When both SPMROOTCR_EL3 and SPMSCR_EL1 are implemented, non-attributable events are counted only if both SPMROOTCR_EL3.NAO is 1 and SPMSCR_EL1.{NAO, SO} is nonzero.

SPMSCR_EL1.NAO has the opposite reset polarity to SPMROOTCR_EL3.NAO.

This field is optional if Root and Realm states are not implemented. When this field is not implemented, System PMU <s> behaves as if SPMSCR_EL1.NAO is 0, and whether events or monitorable characteristics not attributable with any source can be monitored is controlled by SPMSCR_EL1.SO.

The reset behavior of this field is:

- On a System PMU reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bits [3:1]

Reserved, RES0.

SO, bit [0]

Secure Observation. Controls whether events or monitorable characteristics attributable to a Secure event source can be monitored by System PMU <s>.

SO	Meaning
0b0	Events attributable to a Secure event source are not counted by System PMU <s>.
0b1	Counting events by System PMU <s> that are attributable to a Secure event source is not prevented by this mechanism.

Also controls whether events or monitorable characteristics not attributable with any source can be monitored by System PMU <s>. See SPMSCR_EL1.NAO.

The reset behavior of this field is:

- On a System PMU reset, this field resets to '0'.

Accessing SPMSCR_EL1

To access SPMSCR_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMSCR_EL1 reads-as-zero and ignores writes if any of the following are true:

- System PMU <s> is not implemented.
- System PMU <s> does not implement SPMSCR_EL1.

SPMSCR_EL1 is UNDEFINED if accessed in Non-secure or Realm state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMSCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b111	0b1001	0b1110	0b111

```
if !(HaveELUsingSecurityState(EL1, TRUE) && IsFeatureImplemented(FEAT_SPMU) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif IsCurrentSecurityState(SS_NonSecure) || (IsFeatureImplemented(FEAT_RME) &&
IsCurrentSecurityState(SS_Realm)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMSCR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b111	0b1001	0b1110	0b111

```
if !(HaveELUsingSecurityState(EL1, TRUE) && IsFeatureImplemented(FEAT_SPMU) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif IsCurrentSecurityState(SS_NonSecure) || (IsFeatureImplemented(FEAT_RME) &&
IsCurrentSecurityState(SS_Realm)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMSCR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMSELR_EL0, System Performance Monitors Select Register

The SPMSELR_EL0 characteristics are:

Purpose

Selects the System PMU and event counter registers to access.

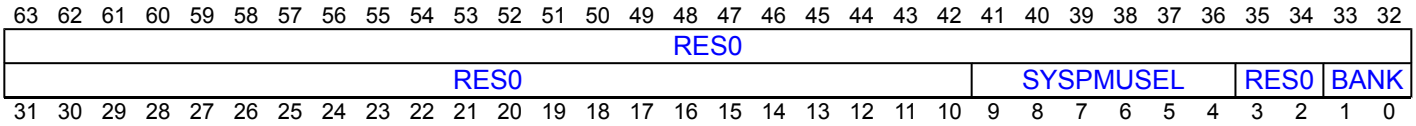
Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMSELR_EL0 are UNDEFINED.

Attributes

SPMSELR_EL0 is a 64-bit register.

Field descriptions



Bits [63:10]

Reserved, RES0.

SYSPMUSEL, bits [9:4]

System PMU Select. Selects a System PMU <s> to access.

Values `0x20` to `0x3F` are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:2]

Reserved, RES0.

BANK, bits [1:0]

System PMU bank access control. Selects a bank of 16 System PMU event counters and related controls to access.

BANK	Meaning
`0b00`	Select event counters 0 to 15.
`0b01`	Select event counters 16 to 31.
`0b10`	Select event counters 32 to 47.
`0b11`	Select event counters 48 to 63.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPMSELR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMSELR_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMSELR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMSELR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nSPMSELR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMSELR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMSELR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMSELR_EL0;
```

```
MSR SPMSELR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMSELR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMSELR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMSELR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPMZR_EL0, System Performance Monitors Zero with Mask

The SPMZR_EL0 characteristics are:

Purpose

Zero the set of System PMU event counters specified by the mask written to SPMZR_EL0.

Configuration

This register is present only when FEAT_SPMU2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMZR_EL0 are UNDEFINED.

Attributes

SPMZR_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

P<m>, bit [m], for m = 63 to 0

Zero event counter <m>.

P<m>	Meaning
0b0	Write is ignored.
0b1	Set event counter<m> in System PMU <s> to zero.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When event counter <m> is not implemented by System PMU <s>, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **WO/RAZ**.

Accessing SPMZR_EL0

To access SPMZR_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMZR_EL0 ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MSR SPMZR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_SPMU2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMZR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMZR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) *
2+:2> != '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMZR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMZR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSel, Stack Pointer Select

The SPSel characteristics are:

Purpose

Allows the Stack Pointer to be selected between SP_EL0 and SP_ELx.

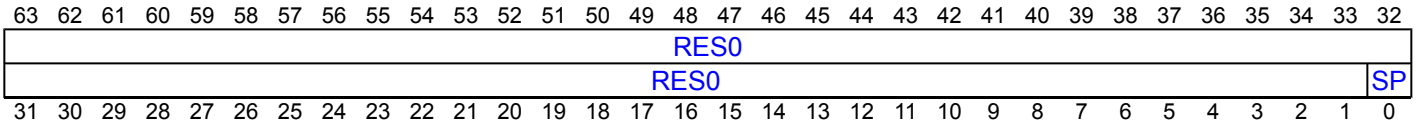
Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SPSel are UNDEFINED.

Attributes

SPSel is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

SP, bit [0]

Stack pointer to use. Possible values of this bit are:

SP	Meaning
0b0	Use SP_EL0 at all Exception levels.
0b1	Use SP_ELx for Exception level ELx. When FEAT_NMI is implemented and SCTLR_ELx.SPINTMASK is 1, if execution is at ELx, an IRQ or FIQ interrupt that is targeted to ELx is masked regardless of any denotion of Superpriority.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

Accessing SPSel

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPSel
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    X[t, 64] = Zeros(63):PSTATE.SP;
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(63):PSTATE.SP;
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(63):PSTATE.SP;
```

MSR SPSel, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    PSTATE.SP = X[t, 64]<0>;
elsif PSTATE.EL == EL2 then
    PSTATE.SP = X[t, 64]<0>;
elsif PSTATE.EL == EL3 then
    PSTATE.SP = X[t, 64]<0>;
```

MSR SPSel, #<imm>

op0	op1	CRn	op2
0b00	0b000	0b0100	0b101

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_abt, Saved Program Status Register (Abort mode)

The SPSR_abt characteristics are:

Purpose

Holds the saved process state when an exception is taken to Abort mode.

Configuration

AArch64 System register SPSR_abt bits [31:0] are architecturally mapped to AArch32 System register SPSR_abt[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SPSR_abt are UNDEFINED.

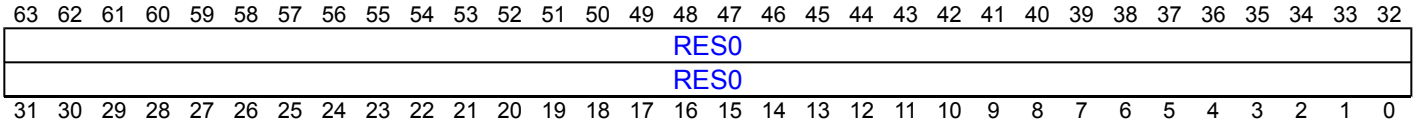
If EL1 only supports execution in AArch64 state, this register is RES0 from EL2 and EL3.

Attributes

SPSR_abt is a 64-bit register.

Field descriptions

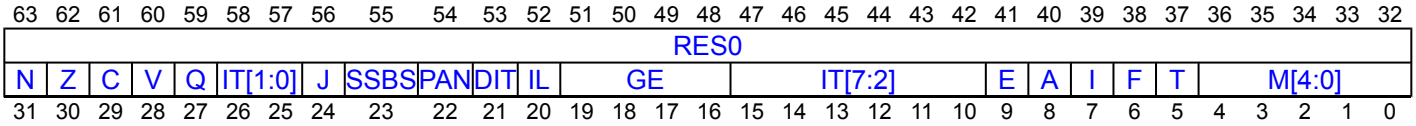
When FEAT_AA32EL1 is not implemented:



Bits [63:0]

Reserved, RES0.

Otherwise:



Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Abort mode, and copied to PSTATE.N on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Abort mode, and copied to PSTATE.Z on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Abort mode, and copied to PSTATE.C on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Abort mode, and copied to PSTATE.V on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Abort mode, and copied to PSTATE.Q on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Abort mode, and copied to PSTATE.IT on executing an exception return operation in Abort mode.

On executing an exception return operation in Abort mode, SPSR_abt.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_abt[26:25].
- IT[7:2] is SPSR_abt[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Abort mode, and copied to PSTATE.SSBS on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Abort mode, and copied to PSTATE.PAN on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Abort mode, and copied to PSTATE.DIT on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Abort mode, and copied to PSTATE.IL on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Abort mode, and copied to PSTATE.GE on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Abort mode, and copied to PSTATE.E on executing an exception return operation in Abort mode.

If the implementation does not support big-endian operation, SPSR_abt.E is RES0. If the implementation does not support little-endian operation, SPSR_abt.E is RES1. On executing an exception return operation in Abort mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_abt.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_abt.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to Abort mode, and copied to PSTATE.A on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Abort mode, and copied to PSTATE.I on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Abort mode, and copied to PSTATE.F on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Abort mode, and copied to PSTATE.T on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Abort mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Abort mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10111	Abort.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_abt.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Abort mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_abt

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPSR_abt
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SPSR_abt;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPSR_abt;
```

```
MSR SPSR_abt, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SPSR_abt = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPSR_abt = X[t, 64];
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_EL1, Saved Program Status Register (EL1)

The SPSR_EL1 characteristics are:

Purpose

Holds the saved process state when an exception is taken to EL1.

Configuration

AArch64 System register SPSR_EL1 bits [31:0] are architecturally mapped to AArch32 System register SPSR_svc[31:0].

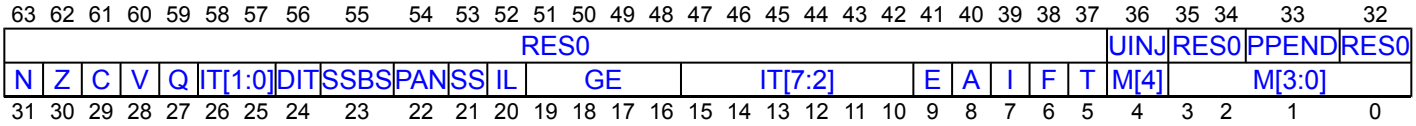
This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SPSR_EL1 are UNDEFINED.

Attributes

SPSR_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented and exception taken from AArch32 state:



An exception return from EL1 using AArch64 makes SPSR_EL1 become UNKNOWN.

Bits [63:37]

Reserved, RES0.

UINJ, bit [36]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Set to 0 on taking an exception to EL1, and copied to PSTATE.UINJ on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:34]

Reserved, RES0.

PPEND, bit [33]

When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Set to the value of PSTATE.PPEND on taking an exception to EL1, and conditionally copied to PSTATE.PPEND on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL1, and copied to PSTATE.N on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL1, and copied to PSTATE.Z on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL1, and copied to PSTATE.C on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL1, and copied to PSTATE.V on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to EL1, and copied to PSTATE.Q on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to EL1, and copied to PSTATE.IT on executing an exception return operation in EL1.

On executing an exception return operation in EL1, SPSR_EL1.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_EL1[26:25].
- IT[7:2] is SPSR_EL1[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL1, and copied to PSTATE.DIT on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL1, and copied to PSTATE.SSBS on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL1, and copied to PSTATE.PAN on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL1, and conditionally copied to PSTATE.SS on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL1, and copied to PSTATE.IL on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to EL1, and copied to PSTATE.GE on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to EL1, and copied to PSTATE.E on executing an exception return operation in EL1.

If the implementation does not support big-endian operation, SPSR_EL1.E is RES0. If the implementation does not support little-endian operation, SPSR_EL1.E is RES1. On executing an exception return operation in EL1, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_EL1.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_EL1.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to EL1, and copied to PSTATE.A on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL1, and copied to PSTATE.I on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL1, and copied to PSTATE.F on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to EL1, and copied to PSTATE.T on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Set to `0b1`, the value of PSTATE.nRW, on taking an exception to EL1 from AArch32 state, and copied to PSTATE.nRW on executing an exception return operation in EL1.

M[4]	Meaning
`0b1`	AArch32 execution state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch32 Mode. Set to the value of PSTATE.M[3:0] on taking an exception to EL1, and copied to PSTATE.M[3:0] on executing an exception return operation in EL1.

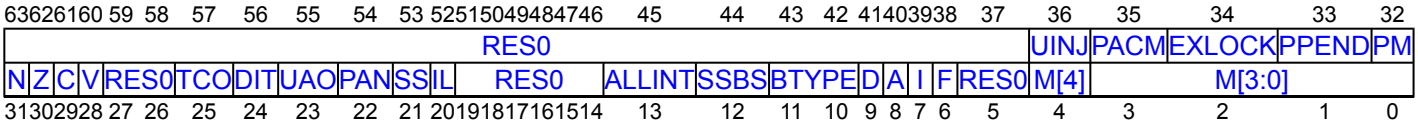
M[3:0]	Meaning
`0b0000`	User.
`0b0001`	FIQ.
`0b0010`	IRQ.
`0b0011`	Supervisor.
`0b0111`	Abort.
`0b1011`	Undefined.
`0b1111`	System.

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL1 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When exception taken from AArch64 state:



An exception return from EL1 using AArch64 makes SPSR_EL1 become UNKNOWN.

Bits [63:37]

Reserved, RES0.

UINJ, bit [36]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Set to 0 on taking an exception to EL1, and copied to PSTATE.UINJ on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PACM, bit [35]
When FEAT_PAuth_LR is implemented:

PACM. Set to the value of PSTATE.PACM on taking an exception to EL1, and copied to PSTATE.PACM on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLOCK, bit [34]
When FEAT_GCS is implemented:

Exception return state lock. Set to the value of PSTATE.EXLOCK on taking an exception to EL1, and copied to PSTATE.EXLOCK on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PPEND, bit [33]
When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Set to the value of PSTATE.PPEND on taking an exception to EL1, and conditionally copied to PSTATE.PPEND on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PM, bit [32]
When FEAT_EBEP is implemented:

Profiling exception mask bit. Set to the value of PSTATE.PM on taking an exception to EL1, and copied to PSTATE.PM on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL1, and copied to PSTATE.N on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL1, and copied to PSTATE.Z on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL1, and copied to PSTATE.C on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL1, and copied to PSTATE.V on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]
When FEAT_MTE is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on taking an exception to EL1, and copied to PSTATE.TCO on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]
When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL1, and copied to PSTATE.DIT on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]
When FEAT_UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on taking an exception to EL1, and copied to PSTATE.UAO on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]
When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL1, and copied to PSTATE.PAN on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL1, and conditionally copied to PSTATE.SS on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL1, and copied to PSTATE.IL on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

ALLINT, bit [13]

When FEAT_NMI is implemented:

All IRQ or FIQ interrupts mask. Set to the value of PSTATE.ALLINT on taking an exception to EL1, and copied to PSTATE.ALLINT on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [12]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL1, and copied to PSTATE.SSBS on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]

When FEAT_BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on taking an exception to EL1, and copied to PSTATE.BTYPE on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on taking an exception to EL1, and copied to PSTATE.D on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to EL1, and copied to PSTATE.A on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL1, and copied to PSTATE.I on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL1, and copied to PSTATE.F on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to `0b0`, the value of PSTATE.nRW, on taking an exception to EL1 from AArch64 state, and copied to PSTATE.nRW on executing an exception return operation in EL1.

M[4]	Meaning
`0b0`	AArch64 execution state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

M[3:0]	Meaning
`0b0000`	EL0.
`0b0100`	EL1 with SP_EL0 (EL1t).
`0b0101`	EL1 with SP_EL1 (EL1h).
`0b1000`	EL1 with SP_EL0 (EL1t) and either HCR_EL2.{NV, NV1} is {1,0} or HCR_EL2.{NV, NV2} is {1,1}.
`0b1001`	EL1 with SP_EL1 (EL1h) and either HCR_EL2.{NV, NV1} is {1,0} or HCR_EL2.{NV, NV2} is {1,1}.

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL1 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The bits in this field are interpreted as follows:

- M[3:2]: On an exception to EL1:
  - If the exception is taken from EL0:
    - M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1.
  - If the exception is taken from EL1:
    - If the Effective value of HCR_EL2.{NV, NV1} is not {1,0} and the Effective value of HCR_EL2.{NV, NV2} is not {1,1}, then M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1.
    - If the Effective value of HCR_EL2.{NV, NV1} is {1,0} or if the Effective value of HCR_EL2.{NV, NV2} is {1,1}, then M[3:2] is set to `0b10`.
  - M[3:2] is copied to PSTATE.EL on executing a legal exception return operation in EL1.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on taking an exception to EL1 and copied to PSTATE.SP on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name SPSR_EL1 or SPSR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x160];
    else
        X[t, 64] = SPSR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = SPSR_EL2;
    else
        X[t, 64] = SPSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPSR_EL1;
```

```
MSR SPSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& !(EffectiveHCR_EL2_NVx() IN {'x11'}) then
        EXLOCKException();
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x160] = X[t, 64];
    else
        SPSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& ELIsInHost(EL2) then
        EXLOCKException();
    elsif ELIsInHost(EL2) then
        SPSR_EL2 = X[t, 64];
    else
        SPSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPSR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, SPSR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x160];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = SPSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SPSR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR SPSR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x160] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        SPSR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SPSR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MRS <Xt>, SPSR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = SPSR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SPSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPSR_EL2;
```

When FEAT_VHE is implemented

```
MSR SPSR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& EffectiveHCR_EL2_NVx() IN {'xx1'} then
        EXLOCKException();
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        SPSR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
then
        EXLOCKException();
    else
        SPSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPSR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_EL2, Saved Program Status Register (EL2)

The SPSR_EL2 characteristics are:

Purpose

Holds the saved process state when an exception is taken to EL2.

Configuration

AArch64 System register SPSR_EL2 bits [31:0] are architecturally mapped to AArch32 System register SPSR_hyp[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SPSR_EL2 are UNDEFINED.

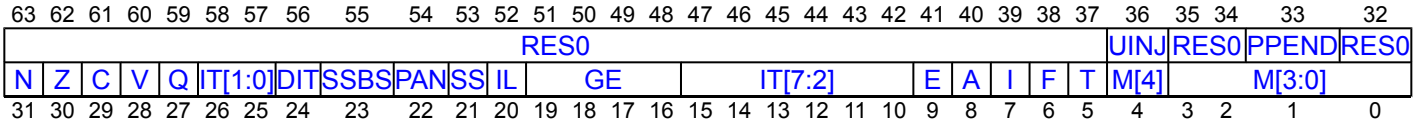
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

SPSR_EL2 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented and exception taken from AArch32 state:



An exception return from EL2 using AArch64 makes SPSR_EL2 become UNKNOWN.

Bits [63:37]

Reserved, RES0.

UINJ, bit [36]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Set to 0 on taking an exception to EL2, and copied to PSTATE.UINJ on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:34]

Reserved, RES0.

PPEND, bit [33]

When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Set to the value of PSTATE.PPEND on taking an exception to EL2, and conditionally copied to PSTATE.PPEND on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL2, and copied to PSTATE.N on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL2, and copied to PSTATE.Z on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL2, and copied to PSTATE.C on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL2, and copied to PSTATE.V on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to EL2, and copied to PSTATE.Q on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to EL2, and copied to PSTATE.IT on executing an exception return operation in EL2.

On executing an exception return operation in EL2, SPSR_EL2.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_EL2[26:25].
- IT[7:2] is SPSR_EL2[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL2, and copied to PSTATE.DIT on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL2, and copied to PSTATE.SSBS on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL2, and copied to PSTATE.PAN on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL2, and conditionally copied to PSTATE.SS on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL2, and copied to PSTATE.IL on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to EL2, and copied to PSTATE.GE on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to EL2, and copied to PSTATE.E on executing an exception return operation in EL2.

If the implementation does not support big-endian operation, SPSR_EL2.E is RES0. If the implementation does not support little-endian operation, SPSR_EL2.E is RES1. On executing an exception return operation in EL2, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_EL2.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_EL2.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to EL2, and copied to PSTATE.A on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL2, and copied to PSTATE.I on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL2, and copied to PSTATE.F on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to EL2, and copied to PSTATE.T on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Set to `0b1`, the value of PSTATE.nRW, on taking an exception to EL2 from AArch32 state, and copied to PSTATE.nRW on executing an exception return operation in EL2.

M[4]	Meaning
`0b1`	AArch32 execution state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch32 Mode. Set to the value of PSTATE.M[3:0] on taking an exception to EL2, and copied to PSTATE.M[3:0] on executing an exception return operation in EL2.

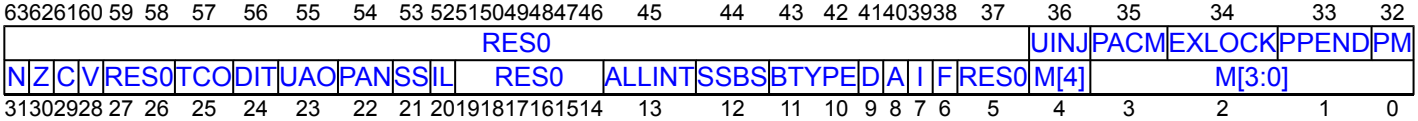
M[3:0]	Meaning
`0b0000`	User.
`0b0001`	FIQ.
`0b0010`	IRQ.
`0b0011`	Supervisor.
`0b0111`	Abort.
`0b1010`	Hyp.
`0b1011`	Undefined.
`0b1111`	System.

Other values are reserved. If SPSR_EL2.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL2 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When exception taken from AArch64 state:



An exception return from EL2 using AArch64 makes SPSR_EL2 become UNKNOWN.

Bits [63:37]

Reserved, RES0.

UINJ, bit [36]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Set to 0 on taking an exception to EL2, and copied to PSTATE.UINJ on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PACM, bit [35]
When FEAT_PAuth_LR is implemented:

PACM. Set to the value of PSTATE.PACM on taking an exception to EL2, and copied to PSTATE.PACM on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLOCK, bit [34]
When FEAT_GCS is implemented:

Exception return state lock. Set to the value of PSTATE.EXLOCK on taking an exception to EL2, and copied to PSTATE.EXLOCK on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PPEND, bit [33]
When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Set to the value of PSTATE.PPEND on taking an exception to EL2, and conditionally copied to PSTATE.PPEND on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PM, bit [32]
When FEAT_EBEP is implemented:

Profiling exception mask bit. Set to the value of PSTATE.PM on taking an exception to EL2, and copied to PSTATE.PM on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL2, and copied to PSTATE.N on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL2, and copied to PSTATE.Z on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL2, and copied to PSTATE.C on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL2, and copied to PSTATE.V on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]
When FEAT_MTE is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on taking an exception to EL2, and copied to PSTATE.TCO on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]
When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL2, and copied to PSTATE.DIT on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]
When FEAT_UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on taking an exception to EL2, and copied to PSTATE.UAO on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]
When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL2, and copied to PSTATE.PAN on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL2, and conditionally copied to PSTATE.SS on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL2, and copied to PSTATE.IL on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

ALLINT, bit [13]

When FEAT_NMI is implemented:

All IRQ or FIQ interrupts mask. Set to the value of PSTATE.ALLINT on taking an exception to EL2, and copied to PSTATE.ALLINT on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [12]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL2, and copied to PSTATE.SSBS on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]

When FEAT_BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on taking an exception to EL2, and copied to PSTATE.BTYPE on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on taking an exception to EL2, and copied to PSTATE.D on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to EL2, and copied to PSTATE.A on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL2, and copied to PSTATE.I on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL2, and copied to PSTATE.F on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to `0b0`, the value of PSTATE.nRW, on taking an exception to EL2 from AArch64 state, and copied to PSTATE.nRW on executing an exception return operation in EL2.

M[4]	Meaning
`0b0`	AArch64 execution state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

M[3:0]	Meaning
`0b0000`	EL0.
`0b0100`	EL1 with SP_EL0 (EL1t).
`0b0101`	EL1 with SP_EL1 (EL1h).
`0b1000`	EL2 with SP_EL0 (EL2t).
`0b1001`	EL2 with SP_EL2 (EL2h).

Other values are reserved. If SPSR_EL2.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL2 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on taking an exception to EL2 and copied to PSTATE.EL on executing an exception return operation in EL2.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on taking an exception to EL2 and copied to PSTATE.SP on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SPSR_EL2 or SPSR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPSR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = SPSR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SPSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPSR_EL2;
```

```
MSR SPSR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& EffectiveHCR_EL2_NVx() IN {'xx1'} then
        EXLOCKException();
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        SPSR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
then
        EXLOCKException();
    else
        SPSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPSR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, SPSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x160];
    else
        X[t, 64] = SPSR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = SPSR_EL2;
    else
        X[t, 64] = SPSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPSR_EL1;
```

When FEAT_VHE is implemented

```
MSR SPSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& !(EffectiveHCR_EL2_NVx() IN {'x11'}) then
        EXLOCKException();
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x160] = X[t, 64];
    else
        SPSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
&& ELIsInHost(EL2) then
        EXLOCKException();
    elsif ELIsInHost(EL2) then
        SPSR_EL2 = X[t, 64];
    else
        SPSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPSR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_EL3, Saved Program Status Register (EL3)

The SPSR_EL3 characteristics are:

Purpose

Holds the saved process state when an exception is taken to EL3.

Configuration

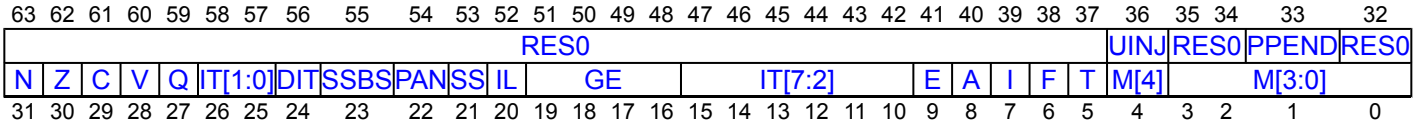
This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPSR_EL3 are UNDEFINED.

Attributes

SPSR_EL3 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented and exception taken from AArch32 state:



An exception return from EL3 using AArch64 makes SPSR_EL3 become UNKNOWN.

Bits [63:37]

Reserved, RES0.

UINJ, bit [36]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Set to 0 on taking an exception to EL3, and copied to PSTATE.UINJ on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:34]

Reserved, RES0.

PPEND, bit [33]

When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Set to the value of PSTATE.PPEND on taking an exception to EL3, and conditionally copied to PSTATE.PPEND on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL3, and copied to PSTATE.N on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL3, and copied to PSTATE.Z on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL3, and copied to PSTATE.C on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL3, and copied to PSTATE.V on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to EL3, and copied to PSTATE.Q on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to EL3, and copied to PSTATE.IT on executing an exception return operation in EL3.

On executing an exception return operation in EL3, SPSR_EL3.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_EL3[26:25].
- IT[7:2] is SPSR_EL3[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL3, and copied to PSTATE.DIT on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL3, and copied to PSTATE.SSBS on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL3, and copied to PSTATE.PAN on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL3, and conditionally copied to PSTATE.SS on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL3, and copied to PSTATE.IL on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to EL3, and copied to PSTATE.GE on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to EL3, and copied to PSTATE.E on executing an exception return operation in EL3.

If the implementation does not support big-endian operation, SPSR_EL3.E is RES0. If the implementation does not support little-endian operation, SPSR_EL3.E is RES1. On executing an exception return operation in EL3, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_EL3.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_EL3.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to EL3, and copied to PSTATE.A on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL3, and copied to PSTATE.I on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL3, and copied to PSTATE.F on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to EL3, and copied to PSTATE.T on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Set to `0b1`, the value of PSTATE.nRW, on taking an exception to EL3 from AArch32 state, and copied to PSTATE.nRW on executing an exception return operation in EL3.

M[4]	Meaning
`0b1`	AArch32 execution state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch32 Mode. Set to the value of PSTATE.M[3:0] on taking an exception to EL3, and copied to PSTATE.M[3:0] on executing an exception return operation in EL3.

M[3:0]	Meaning
`0b0000`	User.
`0b0001`	FIQ.
`0b0010`	IRQ.
`0b0011`	Supervisor.
`0b0110`	Monitor.
`0b0111`	Abort.
`0b1010`	Hyp.
`0b1011`	Undefined.
`0b1111`	System.

Other values are reserved. If SPSR_EL3.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL3 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When exception taken from AArch64 state:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																											UINJ	PACM	EXLOCK	PPEND	PM
N	Z	C	V	RES0		TCO	DIT	UAO	PAN	SS	IL	RES0						ALLINT	SSBS	BTYPE		D	A	I	F	RES0	M[4]	M[3:0]			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

An exception return from EL3 using AArch64 makes SPSR_EL3 become UNKNOWN.

Bits [63:37]

Reserved, RES0.

UINJ, bit [36]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Set to 0 on taking an exception to EL3, and copied to PSTATE.UINJ on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PACM, bit [35]

When FEAT_PAuth_LR is implemented:

PACM. Set to the value of PSTATE.PACM on taking an exception to EL3, and copied to PSTATE.PACM on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLOCK, bit [34]

When FEAT_GCS is implemented:

Exception return state lock. Set to the value of PSTATE.EXLOCK on taking an exception to EL3, and copied to PSTATE.EXLOCK on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PPEND, bit [33]

When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Set to the value of PSTATE.PPEND on taking an exception to EL3, and conditionally copied to PSTATE.PPEND on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PM, bit [32]

When FEAT_EBEP is implemented:

Profiling exception mask bit. Set to the value of PSTATE.PM on taking an exception to EL3, and copied to PSTATE.PM on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL3, and copied to PSTATE.N on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL3, and copied to PSTATE.Z on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL3, and copied to PSTATE.C on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL3, and copied to PSTATE.V on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]

When FEAT_MTE is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on taking an exception to EL3, and copied to PSTATE.TCO on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]
When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL3, and copied to PSTATE.DIT on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]
When FEAT_UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on taking an exception to EL3, and copied to PSTATE.UAO on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]
When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL3, and copied to PSTATE.PAN on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL3, and conditionally copied to PSTATE.SS on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL3, and copied to PSTATE.IL on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

ALLINT, bit [13]

When FEAT_NMI is implemented:

All IRQ or FIQ interrupts mask. Set to the value of PSTATE.ALLINT on taking an exception to EL3, and copied to PSTATE.ALLINT on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [12]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL3, and copied to PSTATE.SSBS on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]

When FEAT_BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on taking an exception to EL3, and copied to PSTATE.BTYPE on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on taking an exception to EL3, and copied to PSTATE.D on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to EL3, and copied to PSTATE.A on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL3, and copied to PSTATE.I on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL3, and copied to PSTATE.F on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to `0b0`, the value of PSTATE.nRW, on taking an exception to EL3 from AArch64 state, and copied to PSTATE.nRW on executing an exception return operation in EL3.

M[4]	Meaning
`0b0`	AArch64 execution state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

M[3:0]	Meaning
`0b0000`	EL0.
`0b0100`	EL1 with SP_EL0 (EL1t).
`0b0101`	EL1 with SP_EL1 (EL1h).
`0b1000`	EL2 with SP_EL0 (EL2t).
`0b1001`	EL2 with SP_EL2 (EL2h).
`0b1100`	EL3 with SP_EL0 (EL3t).
`0b1101`	EL3 with SP_EL3 (EL3h).

Other values are reserved. If SPSR_EL3.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL3 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on taking an exception to EL3 and copied to PSTATE.EL on executing an exception return operation in EL3.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on taking an exception to EL3 and copied to PSTATE.SP on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPSR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0100	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPSR_EL3;
```

```
MSR SPSR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0100	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_GCS) && GetCurrentEXLOCKEN() && !Halted() && PSTATE.EXLOCK == '1'
then
        EXLOCKException();
    else
        SPSR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_fiq, Saved Program Status Register (FIQ mode)

The SPSR_fiq characteristics are:

Purpose

Holds the saved process state when an exception is taken to FIQ mode.

Configuration

AArch64 System register SPSR_fiq bits [31:0] are architecturally mapped to AArch32 System register SPSR_fiq[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SPSR_fiq are UNDEFINED.

If EL1 only supports execution in AArch64 state, this register is RES0 from EL2 and EL3.

Attributes

SPSR_fiq is a 64-bit register.

Field descriptions

When FEAT_AA32EL1 is not implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
RES0																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Reserved, RES0.

Otherwise:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
N	Z	C	V	Q	IT[1:0]		J	SSBS	PAN	DIT	IL	GE				IT[7:2]						E	A	I	F	T	M[4:0]				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to FIQ mode, and copied to PSTATE.N on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to FIQ mode, and copied to PSTATE.Z on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to FIQ mode, and copied to PSTATE.C on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to FIQ mode, and copied to PSTATE.V on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to FIQ mode, and copied to PSTATE.Q on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to FIQ mode, and copied to PSTATE.IT on executing an exception return operation in FIQ mode.

On executing an exception return operation in FIQ mode, SPSR_fiq.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_fiq[26:25].
- IT[7:2] is SPSR_fiq[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to FIQ mode, and copied to PSTATE.SSBS on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to FIQ mode, and copied to PSTATE.PAN on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to FIQ mode, and copied to PSTATE.DIT on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to FIQ mode, and copied to PSTATE.IL on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to FIQ mode, and copied to PSTATE.GE on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to FIQ mode, and copied to PSTATE.E on executing an exception return operation in FIQ mode.

If the implementation does not support big-endian operation, SPSR_fiq.E is RES0. If the implementation does not support little-endian operation, SPSR_fiq.E is RES1. On executing an exception return operation in FIQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_fiq.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_fiq.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to FIQ mode, and copied to PSTATE.A on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to FIQ mode, and copied to PSTATE.I on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to FIQ mode, and copied to PSTATE.F on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to FIQ mode, and copied to PSTATE.T on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to FIQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in FIQ mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10111	Abort.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_fiq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in FIQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_fiq

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPSR_fiq
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0011	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SPSR_fiq;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPSR_fiq;
```

```
MSR SPSR_fiq, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0011	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SPSR_fiq = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPSR_fiq = X[t, 64];
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_irq, Saved Program Status Register (IRQ mode)

The SPSR_irq characteristics are:

Purpose

Holds the saved process state when an exception is taken to IRQ mode.

Configuration

AArch64 System register SPSR_irq bits [31:0] are architecturally mapped to AArch32 System register SPSR_irq[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SPSR_irq are UNDEFINED.

If EL1 only supports execution in AArch64 state, this register is RES0 from EL2 and EL3.

Attributes

SPSR_irq is a 64-bit register.

Field descriptions

When FEAT_AA32EL1 is not implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
RES0																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Reserved, RES0.

Otherwise:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
N	Z	C	V	Q	IT[1:0]		J	SSBS	PAN	DIT	IL	GE				IT[7:2]						E	A	I	F	T	M[4:0]				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to IRQ mode, and copied to PSTATE.N on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to IRQ mode, and copied to PSTATE.Z on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to IRQ mode, and copied to PSTATE.C on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to IRQ mode, and copied to PSTATE.V on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to IRQ mode, and copied to PSTATE.Q on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to IRQ mode, and copied to PSTATE.IT on executing an exception return operation in IRQ mode.

On executing an exception return operation in IRQ mode, SPSR_irq.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_irq[26:25].
- IT[7:2] is SPSR_irq[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to IRQ mode, and copied to PSTATE.SSBS on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to IRQ mode, and copied to PSTATE.PAN on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to IRQ mode, and copied to PSTATE.DIT on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to IRQ mode, and copied to PSTATE.IL on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to IRQ mode, and copied to PSTATE.GE on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to IRQ mode, and copied to PSTATE.E on executing an exception return operation in IRQ mode.

If the implementation does not support big-endian operation, SPSR_irq.E is RES0. If the implementation does not support little-endian operation, SPSR_irq.E is RES1. On executing an exception return operation in IRQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_irq.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_irq.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to IRQ mode, and copied to PSTATE.A on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to IRQ mode, and copied to PSTATE.I on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to IRQ mode, and copied to PSTATE.F on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to IRQ mode, and copied to PSTATE.T on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to IRQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in IRQ mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10111	Abort.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_irq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in IRQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_irq

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPSR_irq
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SPSR_irq;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPSR_irq;
```

```
MSR SPSR_irq, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SPSR_irq = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPSR_irq = X[t, 64];
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_und, Saved Program Status Register (Undefined mode)

The SPSR_und characteristics are:

Purpose

Holds the saved process state when an exception is taken to Undefined mode.

Configuration

AArch64 System register SPSR_und bits [31:0] are architecturally mapped to AArch32 System register SPSR_und[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SPSR_und are UNDEFINED.

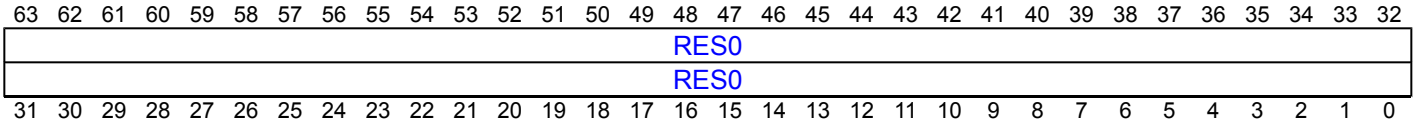
If EL1 only supports execution in AArch64 state, this register is RES0 from EL2 and EL3.

Attributes

SPSR_und is a 64-bit register.

Field descriptions

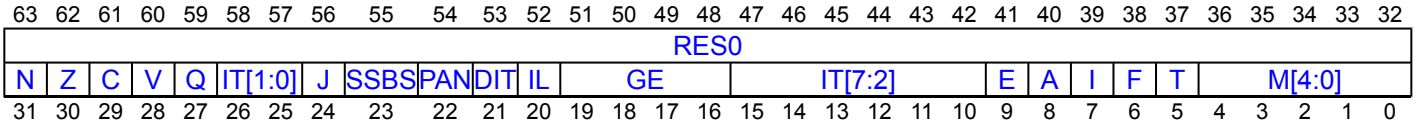
When FEAT_AA32EL1 is not implemented:



Bits [63:0]

Reserved, RES0.

Otherwise:



Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Undefined mode, and copied to PSTATE.N on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Undefined mode, and copied to PSTATE.Z on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Undefined mode, and copied to PSTATE.C on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Undefined mode, and copied to PSTATE.V on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Undefined mode, and copied to PSTATE.Q on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Undefined mode, and copied to PSTATE.IT on executing an exception return operation in Undefined mode.

On executing an exception return operation in Undefined mode, SPSR_und.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_und[26:25].
- IT[7:2] is SPSR_und[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Undefined mode, and copied to PSTATE.SSBS on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Undefined mode, and copied to PSTATE.PAN on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Undefined mode, and copied to PSTATE.DIT on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Undefined mode, and copied to PSTATE.IL on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Undefined mode, and copied to PSTATE.GE on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Undefined mode, and copied to PSTATE.E on executing an exception return operation in Undefined mode.

If the implementation does not support big-endian operation, SPSR_und.E is RES0. If the implementation does not support little-endian operation, SPSR_und.E is RES1. On executing an exception return operation in Undefined mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_und.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_und.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to Undefined mode, and copied to PSTATE.A on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Undefined mode, and copied to PSTATE.I on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Undefined mode, and copied to PSTATE.F on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Undefined mode, and copied to PSTATE.T on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Undefined mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Undefined mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10111	Abort.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_und.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Undefined mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_und

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SPSR_und

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SPSR_und;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPSR_und;
```

MSR SPSR_und, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0100	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SPSR_und = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPSR_und = X[t, 64];
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SSBS, Speculative Store Bypass Safe

The SSBS characteristics are:

Purpose

Allows access to the Speculative Store Bypass Safe bit.

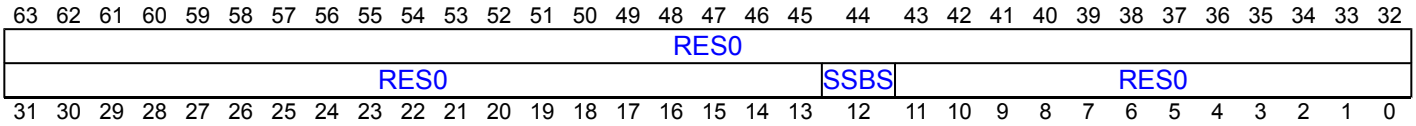
Configuration

This register is present only when FEAT_SSBS2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SSBS are UNDEFINED.

Attributes

SSBS is a 64-bit register.

Field descriptions



Bits [63:13]

Reserved, RES0.

SSBS, bit [12]

Speculative Store Bypass Safe.

Prohibits speculative loads or stores which might practically allow a cache timing side channel.

A speculative value in a register is used in a potentially speculatively exploitable manner if it is used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence or if the execution timing of any other instructions in the speculative sequence is a function of the data loaded under speculation.

SSBS	Meaning
0b0	Hardware is not permitted to use speculative register values in a potentially speculatively exploitable manner if the speculative read that loads the register is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.
0b1	When the value of PSTATE.SSBS is 1, hardware is permitted to use speculative register values in a potentially speculatively exploitable manner if the speculative read that loads the register is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.

The value of this bit is set to the value in the SCTLR_ELx.DSSBS field on taking an exception to ELx.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [11:0]

Reserved, RES0.

Accessing SSBS

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SSBS

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b110

```
if !(IsFeatureImplemented(FEAT_SSBS2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    X[t, 64] = Zeros(51):PSTATE.SSBS:Zeros(12);
elsif PSTATE.EL == EL1 then
    X[t, 64] = Zeros(51):PSTATE.SSBS:Zeros(12);
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(51):PSTATE.SSBS:Zeros(12);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(51):PSTATE.SSBS:Zeros(12);
```

MSR SSBS, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b110

```
if !(IsFeatureImplemented(FEAT_SSBS2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    PSTATE.SSBS = X[t, 64]<12>;
elsif PSTATE.EL == EL1 then
    PSTATE.SSBS = X[t, 64]<12>;
elsif PSTATE.EL == EL2 then
    PSTATE.SSBS = X[t, 64]<12>;
elsif PSTATE.EL == EL3 then
    PSTATE.SSBS = X[t, 64]<12>;
```

MSR SSBS, #<imm>

op0	op1	CRn	op2
0b00	0b011	0b0100	0b001

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SVCR, Streaming Vector Control Register

The SVCR characteristics are:

Purpose

Controls Streaming SVE mode and SME behavior.

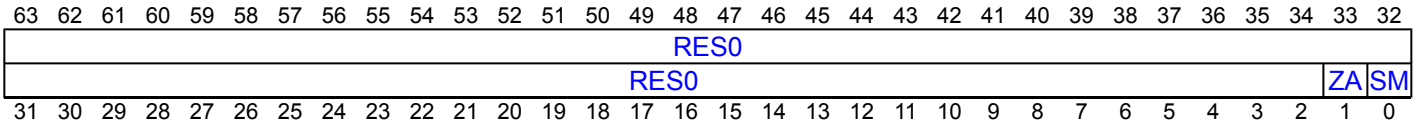
Configuration

This register is present only when FEAT_SME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SVCR are UNDEFINED.

Attributes

SVCR is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

ZA, bit [1]

Enables SME ZA storage. If FEAT_SME2 is implemented, also enables SME2 ZT0 storage.

When this storage is disabled, execution of an instruction which can access it is trapped. The exception is reported using an ESR_ELx.{EC, SMTC} value of {0x1D, 0x3}.

The possible values of this bit are:

ZA	Meaning
0b0	SME ZA storage and, if implemented, ZT0 storage are invalid and not accessible. This control causes execution at any Exception level of instructions that can access this storage to be trapped.
0b1	SME ZA storage and, if implemented, ZT0 storage are valid and accessible. This control does not cause execution of any instructions to be trapped.

When a write to SVCR.ZA changes the value of PSTATE.ZA from 0 to 1, all implemented bits of the storage are set to zero.

Changes to this field do not have an effect on the SVE vector and predicate registers and FPSR.

A direct or indirect read of ZA appears to occur in program order relative to a direct write of SVCR, and to MSR SVCRZA and MSR SVCRSMZA instructions, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

SM, bit [0]

Enables Streaming SVE mode.

When the PE is in Streaming SVE mode, the Streaming SVE vector length (SVL) applies to SVE instructions, and execution at any Exception level of an instruction which is illegal in that mode is trapped. The exception is reported using an ESR_ELx.{EC, SMTC} value of {0x1D, 0x1}.

When the PE is not in Streaming SVE mode, the SVE vector length (VL) applies to SVE instructions, and execution at any Exception level of an instruction which is only legal in that mode is trapped. The exception is reported using an ESR_ELx.{EC, SMTC} value of {0x1D, 0x2}.

The possible values of this bit are:

SM	Meaning
0b0	The PE is not in Streaming SVE mode.
0b1	The PE is in Streaming SVE mode.

When a write to SVCR.SM changes the value of PSTATE.SM, the following applies:

- When changed from 0 to 1, an entry to Streaming SVE mode is performed.
- When changed from 1 to 0, an exit from Streaming SVE mode is performed.
- All implemented bits of the SVE registers Z0-Z31, P0-P15, and FFR in the new mode are set to zero.
- All bits in FPMR are set to zero.
- FPSR in the new mode is set to 0x0000_0000_0800_009f, in which all cumulative status bits are set to 1.

Changes to this field do not have an effect on SME ZA storage or, if implemented, ZT0 storage.

A direct or indirect read of SM appears to occur in program order relative to a direct write of SVCR, and to MSR SVCRSM and MSR SVCRSMZA instructions, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing SVCR

SVCR is read/write and can be accessed from any Exception level.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SVCR
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && CPACR_EL1.SMEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        else
            AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif ELIsInHost(EL0) && CPTR_EL2.SMEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SVCR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SVCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SVCR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SVCR;
```

```
MSR SVCR, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && CPACR_EL1.SMEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        else
            AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif ELIsInHost(EL0) && CPTR_EL2.SMEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SVCR = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SVCR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SVCR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SVCR = X[t, 64];
```

```
MSR SVCRSM, #<imm>
```

op0	op1	CRn	CRm	op2
0b00	0b011	0b0100	0b001x	0b011

```
MSR SVCRZA, #<imm>
```

op0	op1	CRn	CRm	op2
0b00	0b011	0b0100	0b010x	0b011

```
MSR SVCRSMZA, #<imm>
```

op0	op1	CRn	CRm	op2
0b00	0b011	0b0100	0b011x	0b011

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCO, Tag Check Override

The TCO characteristics are:

Purpose

When FEAT_MTE is implemented, this register allows tag checks to be disabled globally.

When FEAT_MTE2 is not implemented, it is IMPLEMENTATION DEFINED if accesses to this register access PSTATE.TCO or are RAZ/WI.

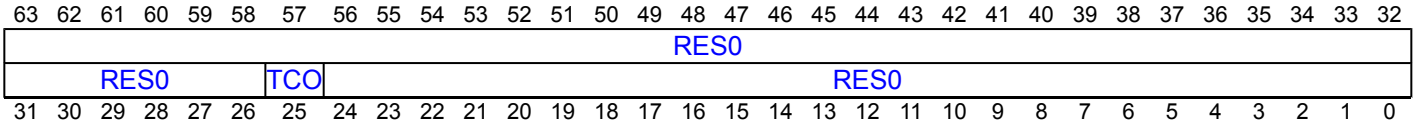
Configuration

This register is present only when FEAT_MTE is implemented. Otherwise, direct accesses to TCO are UNDEFINED.

Attributes

TCO is a 64-bit register.

Field descriptions



Bits [63:26]

Reserved, RES0.

TCO, bit [25]

Allows memory tag checks to be globally disabled.

TCO	Meaning
0b0	Loads and Stores are not affected by this control.
0b1	Loads and Stores are unchecked.

Bits [24:0]

Reserved, RES0.

Accessing TCO

For information about the operation of the MSR (immediate) accessor, see MSR (immediate).

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCO
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b111

```
if !IsFeatureImplemented(FEAT_MTE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    X[t, 64] = Zeros(38):PSTATE.TCO:Zeros(25);
elsif PSTATE.EL == EL1 then
    X[t, 64] = Zeros(38):PSTATE.TCO:Zeros(25);
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(38):PSTATE.TCO:Zeros(25);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(38):PSTATE.TCO:Zeros(25);
```

MSR TCO, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0010	0b111

```
if !IsFeatureImplemented(FEAT_MTE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    PSTATE.TCO = X[t, 64]<25>;
elsif PSTATE.EL == EL1 then
    PSTATE.TCO = X[t, 64]<25>;
elsif PSTATE.EL == EL2 then
    PSTATE.TCO = X[t, 64]<25>;
elsif PSTATE.EL == EL3 then
    PSTATE.TCO = X[t, 64]<25>;
```

MSR TCO, #<imm>

op0	op1	CRn	op2
0b00	0b011	0b0100	0b100

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCR2_EL1, Extended Translation Control Register (EL1)

The TCR2_EL1 characteristics are:

Purpose

The control register for stage 1 of the EL1&0 translation regime.

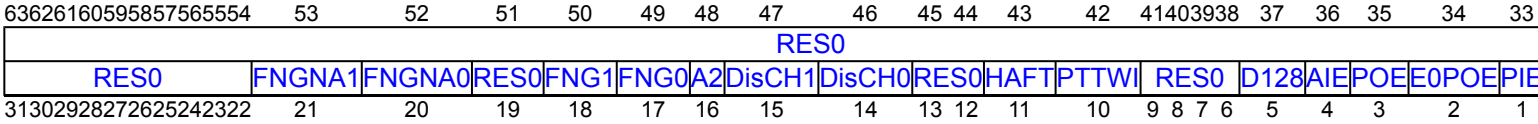
Configuration

This register is present only when FEAT_TCR2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCR2_EL1 are UNDEFINED.

Attributes

TCR2_EL1 is a 64-bit register.

Field descriptions



Unless stated otherwise, all the bits in TCR2_EL1 are permitted to be cached in a TLB.

Bits [63:22]

Reserved, RES0.

FNGNA1, bit [21]

When FEAT_THE is implemented:

Force non-global for unassured translations using TTBR1_EL1.

FNGNA1	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations using TTBR1_EL1 are treated as non-global regardless of the value of the nG bit if all of the following is true: • The translation is for the EL1&0 translation regime. • Stage 1 and stage 2 translation are enabled. • Protection is enabled. • The final stage 1 translation using the descriptor does not have the Assured Translation property.

This bit is permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNGNA0, bit [20]

When FEAT_THE is implemented:

Force non-global for unassured translations using TTBR0_EL1.

FNGNA0	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations using TTBR0_EL1 are treated as non-global regardless of the value of the nG bit if all of the following is true: • The translation is for the EL1&0 translation regime. • Stage 1 and stage 2 translation are enabled. • Protection is enabled. • The final stage 1 translation using the descriptor does not have the Assured Translation property.

This bit is permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [19]

Reserved, RES0.

FNG1, bit [18]

When FEAT_ASID2 is implemented:

Force non-global translations for TTBR1_EL1.

FNG1	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations using TTBR1_EL1 are treated as non-global regardless of the value of the nG bit.

This bit is permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNG0, bit [17]

When FEAT_ASID2 is implemented:

Force non-global translations for TTBR0_EL1.

FNG0	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations using TTBR0_EL1 are treated as non-global regardless of the value of the nG bit.

This bit is permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

A2, bit [16]

When FEAT_ASID2 is implemented:

Enable use of two ASIDs.

A2	Meaning
0b0	Use of two ASIDs is disabled.
0b1	Use of two ASIDs is enabled.

This bit is permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH1, bit [15]

When FEAT_D128 is implemented and TCR2_EL1.D128 == 1:

Disable the Contiguous bit for the Start Table for TTBR1_EL1.

DisCH1	Meaning
0b0	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR1_EL1 is not affected by this field.
0b1	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR1_EL1 is treated as 0.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH0, bit [14]

When FEAT_D128 is implemented and TCR2_EL1.D128 == 1:

Disable the Contiguous bit for the Start Table for TTBR0_EL1.

DisCH0	Meaning
0b0	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR0_EL1 is not affected by this field.
0b1	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR0_EL1 is treated as 0.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [13:12]

Reserved, RES0.

HAFT, bit [11]

When FEAT_HAFT is implemented:

Hardware managed Access Flag for Table descriptors.

Enables the Hardware managed Access Flag for Table descriptors.

HAFT	Meaning
0b0	Hardware managed Access Flag for Table descriptors is disabled.
0b1	Hardware managed Access Flag for Table descriptors is enabled.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

When FEAT_THE is implemented:

Permit Translation table walk Incoherence.

Permits RCWS instructions to generate writes that have the Reduced Coherence property.

PTTWI	Meaning
0b0	Write accesses generated by RCWS at EL1&0 do not have the Reduced Coherence property.
0b1	Write accesses generated by RCWS at EL1&0 have the Reduced Coherence property if HCRX_EL2.PTTWI is 1.

This bit is permitted to be implemented as a read-only bit with a fixed value of 0.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:6]

Reserved, RES0.

D128, bit [5]

When FEAT_D128 is implemented:

Enables VMSAv9-128 translation system.

D128	Meaning
0b0	Translation system follows VMSAv8-64 translation process.
0b1	Translation system follows VMSAv9-128 translation process.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIE, bit [4]

When FEAT_AIE is implemented:

Enable Attribute Indexing Extension.

AIE	Meaning
0b0	Attribute Indexing Extension Disabled.
0b1	Attribute Indexing Extension Enabled.

This field is RES1 when TCR2_EL1.D128 is 1.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]

When FEAT_S1POE is implemented:

Enables Permission Overlays for privileged accesses from EL1&0 translation regime.

POE	Meaning
0b0	Permission overlay disabled for EL1 access in stage 1 of EL1&0 translation regime.
0b1	Permission overlay enabled for EL1 access in stage 1 of EL1&0 translation regime.

This bit is not permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [2]

When FEAT_S1POE is implemented:

Enables Permission Overlays for unprivileged accesses from EL1&0 translation regime.

E0POE	Meaning
0b0	Permission overlay disabled for EL0 access in stage 1 of EL1&0 translation regime.
0b1	Permission overlay enabled for EL0 access in stage 1 of EL1&0 translation regime.

This bit is not permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIE, bit [1]

When FEAT_S1PIE is implemented:

Enables usage of Indirect Permission Scheme.

PIE	Meaning
0b0	Direct permission model.
0b1	Indirect permission model.

This field is RES1 when TCR2_EL1.D128 is 1.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]
When FEAT_THE is implemented:

Protected attribute enable. Enables use of bit[52] of the stage 1 translation table entries as the Protected bit, for translations using TTBRn_EL1.

PnCH	Meaning
0b0	For translations using TTBRn_EL1, bit[52] of each stage 1 translation table entry is not the Protected bit.
0b1	For translations using TTBRn_EL1, bit[52] of each stage 1 translation table entry is the Protected bit.

If bit[52] is used as the Protected bit, it is not used as the Contiguous bit.

This field is RES0 when TCR2_EL1.D128 is 1.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TCR2_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TCR2_EL1 or TCR2_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to TCR2_EL1 are masked by TCR2MASK_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TCR2_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x270];
    else
        X[t, 64] = TCR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCR2_EL2;
    else
        X[t, 64] = TCR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2_EL1;
```

MSR TCR2_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x270] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR2_EL1 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL1()) OR (TCR2_EL1 AND
EffectiveTCR2MASK_EL1());
        else
            TCR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR2_EL2 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL2()) OR (TCR2_EL2 AND
EffectiveTCR2MASK_EL2());
        else
            TCR2_EL2 = X[t, 64];
    else
        TCR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2_EL1 = X[t, 64];
```

MRS <Xt>, TCR2_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x270];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TCR2_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR2_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR TCR2_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x270] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TCR2_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TCR2_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_SRMASK is implemented

```
MRS <Xt>, TCR2ALIAS_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b111

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nTCR2ALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x270];
    else
        X[t, 64] = TCR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCR2_EL2;
    else
        X[t, 64] = TCR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2_EL1;
```

When FEAT_SRMASK is implemented

```
MSR TCR2ALIAS_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b111

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nTCR2ALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x270] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR2_EL1 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL1()) OR (TCR2_EL1 AND
EffectiveTCR2MASK_EL1());
        else
            TCR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR2_EL2 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL2()) OR (TCR2_EL2 AND
EffectiveTCR2MASK_EL2());
        else
            TCR2_EL2 = X[t, 64];
    else
        TCR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCR2_EL2, Extended Translation Control Register (EL2)

The TCR2_EL2 characteristics are:

Purpose

The control register for stage 1 of the EL2&0 translation regime.

Configuration

This register is present only when FEAT_TCR2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCR2_EL2 are UNDEFINED.

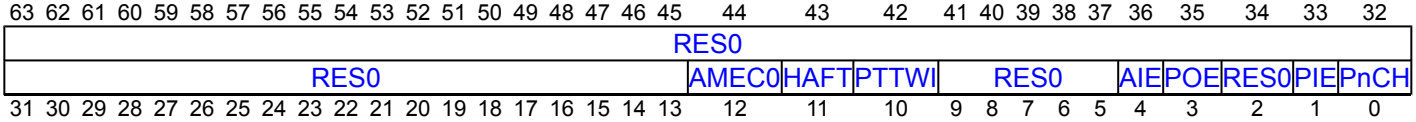
If EL2 is not implemented, this register is RES0 from EL3.

Attributes

TCR2_EL2 is a 64-bit register.

Field descriptions

When !ELIsInHost(EL2):



Unless stated otherwise, all the bits in TCR2_EL2 are permitted to be cached in a TLB.

Bits [63:13]

Reserved, RES0.

AMEC0, bit [12]

When FEAT_MEC is implemented:

This field controls the enabling of the Alternate MECID translations for the EL2 translation regime.

TCR2_EL2.AMEC0 is provided to enable the safe update of MECID_A0_EL2, by disabling access and speculation to AMEC==1 Block or Page descriptors during the update.

AMEC0	Meaning
0b0	Use of a Block or Page descriptor containing AMEC == 1 generates a Translation fault.
0b1	Accesses translated by a Block or Page descriptor containing AMEC == 1 are associated with the MECID configured in MECID_A0_EL2.

This bit is permitted to be cached in a TLB only if it is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

When SCTLR2_EL2.EMEC is 0, this field is ignored by the PE and the bit position of AMEC is RES0 in Block and Page descriptors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When IsCurrentSecurityState(SS_Secure), access to this field is **RES0**.
- When IsCurrentSecurityState(SS_NonSecure), access to this field is **RES0**.

Otherwise:

Reserved, RES0.

HAFT, bit [11]
When FEAT_HAFT is implemented:

Hardware managed Access Flag for Table descriptors.

Enables the Hardware managed Access Flag for Table descriptors.

HAFT	Meaning
0b0	Hardware managed Access Flag for Table descriptors is disabled.
0b1	Hardware managed Access Flag for Table descriptors is enabled.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]
When FEAT_THE is implemented:

Permit Translation table walk Incoherence.

Permits RCWS instructions to generate writes that have the Reduced Coherence property.

PTTWI	Meaning
0b0	Write accesses generated by RCWS at EL2 or EL2&0 do not have the Reduced Coherence property.
0b1	Write accesses generated by RCWS at EL2 or EL2&0 have the Reduced Coherence property.

This bit is permitted to be implemented as a read-only bit with a fixed value of 0.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:5]

Reserved, RES0.

AIE, bit [4]

When FEAT_AIE is implemented:

Enable Attribute Indexing Extension.

AIE	Meaning
0b0	Attribute Indexing Extension Disabled.
0b1	Attribute Indexing Extension Enabled.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]

When FEAT_S1POE is implemented:

Enables Permission Overlay for EL2 accesses.

POE	Meaning
0b0	Permission overlay disabled for EL2 access in stage 1 of EL2 translation regime.
0b1	Permission overlay enabled for EL2 access in stage 1 of EL2 translation regime.

This bit is not permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [2]

Reserved, RES0.

PIE, bit [1]

When FEAT_S1PIE is implemented:

Enables usage of Indirect Permission Scheme.

PIE	Meaning
0b0	Direct permission model.
0b1	Indirect permission model.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]
When FEAT_THE is implemented:

Protected attribute enable. Enables use of bit[52] of stage 1 translation table entries as the Protected bit, for translations using TTBR0_EL2 when HCR_EL2.E2H is 0.

PnCH	Meaning
0b0	For translations using TTBRn_EL2, bit[52] of each stage 1 translation table entry is not the Protected bit.
0b1	For translations using TTBR0_EL2, bit[52] of each stage 1 translation table entry is the Protected bit.

If bit[52] is used as the Protected bit, it is not used as the Contiguous bit.

This field is RES0 when TCR2_EL2.D128 is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When ELIsInHost(EL2):

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
RES0													FNG1	FNG0	A2	DisCH1	DisCH0	AMEC1	AMEC0	HAFT	PTTWI	RES0				D128	AIE	POE	E0POE	PIE	PnCH
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Unless stated otherwise, all the bits in TCR2_EL2 are permitted to be cached in a TLB.

Bits [63:19]

Reserved, RES0.

FNG1, bit [18]
When FEAT_ASID2 is implemented:

Force non-global translations for TTBR1_EL2.

FNG1	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations are treated as non-global regardless of the value of the nG bit.

This bit is permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNG0, bit [17]
When FEAT_ASID2 is implemented:

Force non-global translations for TTBR0_EL2.

FNG0	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations are treated as non-global regardless of the value of the nG bit.

This bit is permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

A2, bit [16]
When FEAT_ASID2 is implemented:

Enable use of two ASIDs.

A2	Meaning
0b0	Use of two ASIDs is disabled.
0b1	Use of two ASIDs is enabled.

This bit is permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH1, bit [15]
When FEAT_D128 is implemented and TCR2_EL2.D128 == 1:

Disable the Contiguous bit for the Start Table for TTBR1_EL2.

DisCH1	Meaning
0b0	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR1_EL2 is not affected by this field.
0b1	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR1_EL2 is treated as 0.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH0, bit [14]

When FEAT_D128 is implemented and TCR2_EL2.D128 == 1:

Disable the Contiguous bit for the Start Table for TTBR0_EL2.

DisCH0	Meaning
0b0	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR0_EL2 is not affected by this field.
0b1	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR0_EL2 is treated as 0.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMEC1, bit [13]

When FEAT_MEC is implemented:

This field controls the enabling of the Alternate MECID translations for accesses in the TTBR1_EL2 half of the VA range, for the EL2&0 translation regime.

TCR2_EL2.AMEC1 is provided to enable the safe update of MECID_A1_EL2, by disabling access and speculation to AMEC == 1 Block or Page descriptors during the update.

AMEC1	Meaning
0b0	Use of a Block or Page descriptor containing AMEC == 1 generates a Translation fault.
0b1	Accesses translated by a Block or Page descriptor containing AMEC == 1 are associated with the MECID configured in MECID_A1_EL2.

This bit is permitted to be cached in a TLB only if it is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

When SCTLR2_EL2.EMEC is 0, this field is ignored by the PE and the bit position of AMEC is RES0 in Block and Page descriptors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When IsCurrentSecurityState(SS_Secure), access to this field is **RES0**.
- When IsCurrentSecurityState(SS_NonSecure), access to this field is **RES0**.

Otherwise:

Reserved, RES0.

AMEC0, bit [12]

When FEAT_MEC is implemented:

This field controls the enabling of the Alternate MECID translations for accesses in the TTBR0_EL2 half of the VA range, for the EL2&0 translation regime.

TCR2_EL2.AMEC0 is provided to enable the safe update of MECID_A0_EL2, by disabling access and speculation to AMEC==1 Block or Page descriptors during the update.

AMEC0	Meaning
0b0	Use of a Block or Page descriptor containing AMEC == 1 generates a Translation fault.
0b1	Accesses translated by a Block or Page descriptor containing AMEC == 1 are associated with the MECID configured in MECID_A0_EL2.

This bit is permitted to be cached in a TLB only if it is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

When SCTLR2_EL2.EMEC is 0, this field is ignored by the PE and the bit position of AMEC is RES0 in Block and Page descriptors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When IsCurrentSecurityState(SS_Secure), access to this field is **RES0**.
- When IsCurrentSecurityState(SS_NonSecure), access to this field is **RES0**.

Otherwise:

Reserved, RES0.

HAFT, bit [11]

When FEAT_HAFT is implemented:

Hardware managed Access Flag for Table descriptors.

Enables the Hardware managed Access Flag for Table descriptors.

HAFT	Meaning
0b0	Hardware managed Access Flag for Table descriptors is disabled.
0b1	Hardware managed Access Flag for Table descriptors is enabled.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

When FEAT_THE is implemented:

Permit Translation table walk Incoherence.

Permits RCWS instructions to generate writes that have the Reduced Coherence property.

PTTWI	Meaning
0b0	Write accesses generated by RCWS do not have the Reduced Coherence property.
0b1	Write accesses generated by RCWS have the Reduced Coherence property.

This bit is permitted to be implemented as a read-only bit with a fixed value of 0.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:6]

Reserved, RES0.

D128, bit [5]

When FEAT_D128 is implemented:

Enables VMSAv9-128 translation system.

D128	Meaning
0b0	Translation system follows VMSAv8-64 translation process.
0b1	Translation system follows VMSAv9-128 translation process.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIE, bit [4]

When FEAT_AIE is implemented:

Enable Attribute Indexing Extension.

AIE	Meaning
0b0	Attribute Indexing Extension Disabled.
0b1	Attribute Indexing Extension Enabled.

This field is RES1 when TCR2_EL2.D128 is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]
When FEAT_S1POE is implemented:

Enables Permission Overlay for privileged accesses from EL2&0 translation regime.

POE	Meaning
0b0	Permission overlay disabled for EL2 access in stage 1 of EL2&0 translation regime.
0b1	Permission overlay enabled for EL2 access in stage 1 of EL2&0 translation regime.

This bit is not permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [2]
When FEAT_S1POE is implemented:

Enables Permission Overlay for unprivileged accesses from EL2&0 translation regime.

E0POE	Meaning
0b0	Permission overlay disabled for EL0 access in stage 1 of EL2&0 translation regime.
0b1	Permission overlay enabled for EL0 access in stage 1 of EL2&0 translation regime.

This bit is not permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIE, bit [1]
When FEAT_S1PIE is implemented:

Enables usage of Indirect Permission Scheme.

PIE	Meaning
0b0	Direct permission model.
0b1	Indirect permission model.

This field is RES1 when TCR2_EL2.D128 is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]
When FEAT_THE is implemented:

Protected attribute enable. Enables use of bit[52] of stage 1 translation table entries as the Protected bit, for translations using TTBRn_EL2 when HCR_EL2.E2H is 1.

PnCH	Meaning
0b0	For translations using TTBRn_EL2, bit[52] of each stage 1 translation table entry is not the Protected bit.
0b1	For translations using TTBR0_EL2, bit[52] of each stage 1 translation table entry is the Protected bit.

If bit[52] is used as the Protected bit, it is not used as the Contiguous bit.

This field is RES0 when TCR2_EL2.D128 is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TCR2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TCR2_EL2 or TCR2_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to TCR2_EL2 are masked by TCR2MASK_EL2.

Accesses to this register use the following encodings in the System register encoding space:

When FEAT_VHE is implemented

```
MRS <Xt>, TCR2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TCR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2_EL2;
```

```
MSR TCR2_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR2_EL2 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL2()) OR (TCR2_EL2 AND
EffectiveTCR2MASK_EL2());
        else
            TCR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2_EL2 = X[t, 64];
```

MRS <Xt>, TCR2_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x270];
    else
        X[t, 64] = TCR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCR2_EL2;
    else
        X[t, 64] = TCR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2_EL1;
```

MSR TCR2_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x270] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR2_EL1 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL1()) OR (TCR2_EL1 AND
EffectiveTCR2MASK_EL1());
        else
            TCR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR2_EL2 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL2()) OR (TCR2_EL2 AND
EffectiveTCR2MASK_EL2());
        else
            TCR2_EL2 = X[t, 64];
    else
        TCR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCR2MASK_EL1, Extended Translation Control Masking Register (EL1)

The TCR2MASK_EL1 characteristics are:

Purpose

Mask register to prevent updates of fields in TCR2_EL1 on writes to TCR2_EL1 or TCR2ALIAS_EL1.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCR2MASK_EL1 are UNDEFINED.

Attributes

TCR2MASK_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
RES0																														
RES0										FNGNA1	FNGNA0	RES0		FNG1	FNG0	A2	DisCH1	DisCH0	RES0		HAFT	PTTWI	RES0			D128	AIE	POE	E0POE	PIE
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Bits [63:22]

Reserved, RES0.

FNGNA1, bit [21]

When FEAT_THE is implemented:

Mask bit for FNGNA1.

FNGNA1	Meaning
0b0	TCR2_EL1.FNGNA1 is writeable.
0b1	TCR2_EL1.FNGNA1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNGNA0, bit [20]

When FEAT_THE is implemented:

Mask bit for FNGNA0.

FNGNA0	Meaning
0b0	TCR2_EL1.FNGNA0 is writeable.
0b1	TCR2_EL1.FNGNA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [19]

Reserved, RES0.

FNG1, bit [18]

When FEAT_ASID2 is implemented:

Mask bit for FNG1.

FNG1	Meaning
0b0	TCR2_EL1.FNG1 is writeable.
0b1	TCR2_EL1.FNG1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNG0, bit [17]

When FEAT_ASID2 is implemented:

Mask bit for FNG0.

FNG0	Meaning
0b0	TCR2_EL1.FNG0 is writeable.
0b1	TCR2_EL1.FNG0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

A2, bit [16]

When FEAT_ASID2 is implemented:

Mask bit for A2.

A2	Meaning
0b0	TCR2_EL1.A2 is writeable.
0b1	TCR2_EL1.A2 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH1, bit [15]

When FEAT_D128 is implemented:

Mask bit for DisCH1.

DisCH1	Meaning
0b0	TCR2_EL1.DisCH1 is writeable.
0b1	TCR2_EL1.DisCH1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH0, bit [14]

When FEAT_D128 is implemented:

Mask bit for DisCH0.

DisCH0	Meaning
0b0	TCR2_EL1.DisCH0 is writeable.
0b1	TCR2_EL1.DisCH0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [13:12]

Reserved, RES0.

HAFT, bit [11]

When FEAT_HAFT is implemented:

Mask bit for HAFT.

HAFT	Meaning
0b0	TCR2_EL1.HAFT is writeable.
0b1	TCR2_EL1.HAFT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

When FEAT_THE is implemented:

Mask bit for PTTWI.

PTTWI	Meaning
0b0	TCR2_EL1.PTTWI is writeable.
0b1	TCR2_EL1.PTTWI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:6]

Reserved, RES0.

D128, bit [5]

When FEAT_D128 is implemented:

Mask bit for D128.

D128	Meaning
0b0	TCR2_EL1.D128 is writeable.
0b1	TCR2_EL1.D128 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIE, bit [4]
When FEAT_AIE is implemented:

Mask bit for AIE.

AIE	Meaning
0b0	TCR2_EL1.AIE is writeable.
0b1	TCR2_EL1.AIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]
When FEAT_S1POE is implemented:

Mask bit for POE.

POE	Meaning
0b0	TCR2_EL1.POE is writeable.
0b1	TCR2_EL1.POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [2]
When FEAT_S1POE is implemented:

Mask bit for E0POE.

E0POE	Meaning
0b0	TCR2_EL1.E0POE is writeable.
0b1	TCR2_EL1.E0POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIE, bit [1]
When FEAT_S1PIE is implemented:

Mask bit for PIE.

PIE	Meaning
0b0	TCR2_EL1.PIE is writeable.
0b1	TCR2_EL1.PIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]
When FEAT_THE is implemented:

Mask bit for PnCH.

PnCH	Meaning
0b0	TCR2_EL1.PnCH is writeable.
0b1	TCR2_EL1.PnCH is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TCR2MASK_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TCR2MASK_EL1 or TCR2MASK_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR2MASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nTCR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x338];
    else
        X[t, 64] = TCR2MASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCR2MASK_EL2;
    else
        X[t, 64] = TCR2MASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2MASK_EL1;
```

```
MSR TCR2MASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nTCR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x338] = X[t, 64];
    elsif !IsZero(EffectiveTCR2MASK_EL1()) then
        UNDEFINED;
    else
        TCR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveTCR2MASK_EL2()) then
            UNDEFINED;
        else
            TCR2MASK_EL2 = X[t, 64];
    else
        TCR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2MASK_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, TCR2MASK_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x338];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TCR2MASK_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR2MASK_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR TCR2MASK_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x338] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TCR2MASK_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TCR2MASK_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCR2MASK_EL2, Extended Translation Control Masking Register (EL2)

The TCR2MASK_EL2 characteristics are:

Purpose

Mask register to prevent updates of fields in TCR2_EL2 on writes.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCR2MASK_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

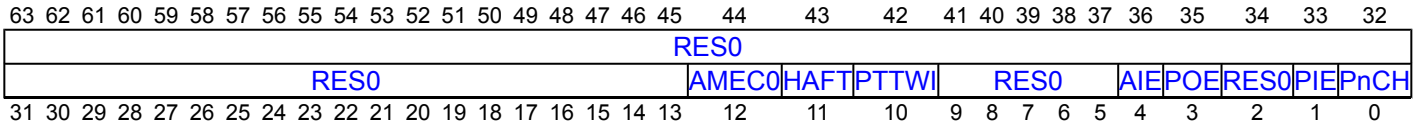
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

TCR2MASK_EL2 is a 64-bit register.

Field descriptions

When !ELIsInHost(EL2):



Bits [63:13]

Reserved, RES0.

AMEC0, bit [12]

When FEAT_MEC is implemented:

Mask bit for AMEC0.

AMEC0	Meaning
0b0	TCR2_EL2.AMEC0 is writeable.
0b1	TCR2_EL2.AMEC0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HAFT, bit [11]

When FEAT_HAFT is implemented:

Mask bit for HAFT.

HAFT	Meaning
0b0	TCR2_EL2.HAFT is writeable.
0b1	TCR2_EL2.HAFT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

When FEAT_THE is implemented:

Mask bit for PTTWI.

PTTWI	Meaning
0b0	TCR2_EL2.PTTWI is writeable.
0b1	TCR2_EL2.PTTWI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:5]

Reserved, RES0.

AIE, bit [4]

When FEAT_AIE is implemented:

Mask bit for AIE.

AIE	Meaning
0b0	TCR2_EL2.AIE is writeable.
0b1	TCR2_EL2.AIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]

When FEAT_S1POE is implemented:

Mask bit for POE.

POE	Meaning
0b0	TCR2_EL2.POE is writeable.
0b1	TCR2_EL2.POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [2]

Reserved, RES0.

PIE, bit [1]

When FEAT_S1PIE is implemented:

Mask bit for PIE.

PIE	Meaning
0b0	TCR2_EL2.PIE is writeable.
0b1	TCR2_EL2.PIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]

When FEAT_THE is implemented:

Mask bit for PnCH.

PnCH	Meaning
0b0	TCR2_EL2.PnCH is writeable.
0b1	TCR2_EL2.PnCH is not writeable.

The reset behavior of this field is:

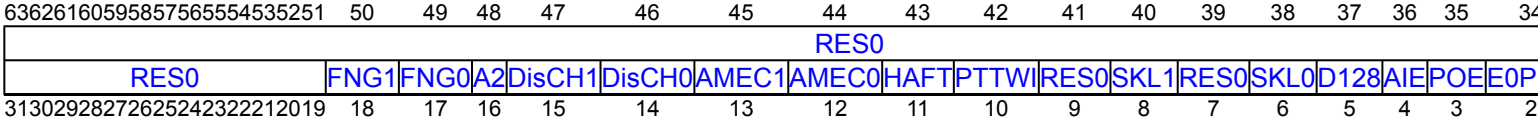
- On a Warm reset:

- When the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When ELIsInHost(EL2):



Bits [63:19]

Reserved, RES0.

FNG1, bit [18]

When FEAT_ASID2 is implemented:

Mask bit for FNG1.

FNG1	Meaning
0b0	TCR2_EL2.FNG1 is writeable.
0b1	TCR2_EL2.FNG1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNG0, bit [17]

When FEAT_ASID2 is implemented:

Mask bit for FNG0.

FNG0	Meaning
0b0	TCR2_EL2.FNG0 is writeable.
0b1	TCR2_EL2.FNG0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

A2, bit [16]

When FEAT_ASID2 is implemented:

Mask bit for A2.

A2	Meaning
0b0	TCR2_EL2.A2 is writeable.
0b1	TCR2_EL2.A2 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH1, bit [15]

When FEAT_D128 is implemented:

Mask bit for DisCH1.

DisCH1	Meaning
0b0	TCR2_EL2.DisCH1 is writeable.
0b1	TCR2_EL2.DisCH1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH0, bit [14]

When FEAT_D128 is implemented:

Mask bit for DisCH0.

DisCH0	Meaning
0b0	TCR2_EL2.DisCH0 is writeable.
0b1	TCR2_EL2.DisCH0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMEC1, bit [13]

When FEAT_MEC is implemented:

Mask bit for AMEC1.

AMEC1	Meaning
0b0	TCR2_EL2.AMEC1 is writeable.
0b1	TCR2_EL2.AMEC1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMEC0, bit [12]

When FEAT_MEC is implemented:

Mask bit for AMEC0.

AMEC0	Meaning
0b0	TCR2_EL2.AMEC0 is writeable.
0b1	TCR2_EL2.AMEC0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HAFT, bit [11]

When FEAT_HAFT is implemented:

Mask bit for HAFT.

HAFT	Meaning
0b0	TCR2_EL2.HAFT is writeable.
0b1	TCR2_EL2.HAFT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

When FEAT_THE is implemented:

Mask bit for PTTWI.

PTTWI	Meaning
0b0	TCR2_EL2.PTTWI is writeable.
0b1	TCR2_EL2.PTTWI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [9]

Reserved, RES0.

SKL1, bit [8]

When FEAT_D128 is implemented:

Mask bit for SKL1.

SKL1	Meaning
0b0	TCR2_EL2.SKL1 is writeable.
0b1	TCR2_EL2.SKL1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

SKL0, bit [6]

When FEAT_D128 is implemented:

Mask bit for SKL0.

SKL0	Meaning
0b0	TCR2_EL2.SKL0 is writeable.
0b1	TCR2_EL2.SKL0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D128, bit [5]
When FEAT_D128 is implemented:

Mask bit for D128.

D128	Meaning
0b0	TCR2_EL2.D128 is writeable.
0b1	TCR2_EL2.D128 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIE, bit [4]
When FEAT_AIE is implemented:

Mask bit for AIE.

AIE	Meaning
0b0	TCR2_EL2.AIE is writeable.
0b1	TCR2_EL2.AIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]
When FEAT_S1POE is implemented:

Mask bit for POE.

POE	Meaning
0b0	TCR2_EL2.POE is writeable.
0b1	TCR2_EL2.POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [2]
When FEAT_S1POE is implemented:

Mask bit for E0POE.

E0POE	Meaning
0b0	TCR2_EL2.E0POE is writeable.
0b1	TCR2_EL2.E0POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIE, bit [1]
When FEAT_S1PIE is implemented:

Mask bit for PIE.

PIE	Meaning
0b0	TCR2_EL2.PIE is writeable.
0b1	TCR2_EL2.PIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]
When FEAT_THE is implemented:

Mask bit for PnCH.

PnCH	Meaning
0b0	TCR2_EL2.PnCH is writeable.
0b1	TCR2_EL2.PnCH is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TCR2MASK_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TCR2MASK_EL2 or TCR2MASK_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR2MASK_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TCR2MASK_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2MASK_EL2;
```

```
MSR TCR2MASK_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsZero(EffectiveTCR2MASK_EL2()) then
        UNDEFINED;
    else
        TCR2MASK_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2MASK_EL2 = X[t, 64];
```

MRS <Xt>, TCR2MASK_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nTCR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x338];
    else
        X[t, 64] = TCR2MASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCR2MASK_EL2;
    else
        X[t, 64] = TCR2MASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2MASK_EL1;
```

```
MSR TCR2MASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nTCR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x338] = X[t, 64];
    elsif !IsZero(EffectiveTCR2MASK_EL1()) then
        UNDEFINED;
    else
        TCR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveTCR2MASK_EL2()) then
            UNDEFINED;
        else
            TCR2MASK_EL2 = X[t, 64];
    else
        TCR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2MASK_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCR_EL1, Translation Control Register (EL1)

The TCR_EL1 characteristics are:

Purpose

The control register for stage 1 of the EL1&0 translation regime.

Configuration

AArch64 System register TCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TTBCR[31:0].

AArch64 System register TCR_EL1 bits [63:32] are architecturally mapped to AArch32 System register TTBCR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TCR_EL1 are UNDEFINED.

Attributes

TCR_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45
RES0		MTX1	MTX0	DS	TCMA1	TCMA0	E0PD1	E0PD0	NFD1	NFD0	TBID1	TBID0	HWU162	HWU161	HWU160	HWU159	HWU062	HWU
TG1		SH1		ORGN1		IRGN1		EPD1	A1	T1SZ						TG0		
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13

Any of the bits in TCR_EL1, other than the EPDx bits when they have the value 1, and the A1 bit are permitted to be cached in a TLB.

Bits [63:62]

Reserved, RES0.

MTX1, bit [61]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL0 and EL1 are using AArch64 where the data address would be translated by tables pointed to by TTBR1_EL1.

This control has an effect regardless of whether stage 1 of the EL1&0 translation regime is enabled or not.

MTX1	Meaning
0b0	This control has no effect on the PE.
0b1	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as 0b1111 when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTX0, bit [60]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL0 and EL1 are using AArch64 where the data address would be translated by tables pointed to by TTBR0_EL1.

This control has an effect regardless of whether stage 1 of the EL1&0 translation regime is enabled or not.

MTX0	Meaning
0b0	This control has no effect on the PE.
0b1	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as 0b0000 when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [59]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or TCR2_EL1.D128 == 0):

This field affects:

- Whether a 52-bit output address can be described by the translation tables of the 4KB or 16KB translation granules.
- The minimum value of TCR_EL1.{T0SZ,T1SZ}.
- How and where shareability for Block and Page descriptors are encoded.

DS	Meaning
0b0	Bits[49:48] of translation descriptors are RES0. Bits[9:8] in Block and Page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in Table descriptors are ignored by hardware. The minimum value of the TCR_EL1.{T0SZ, T1SZ} fields is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. Output address[51:48] is 0b0000.
0b1	Bits[49:48] of translation descriptors hold output address[49:48]. Bits[9:8] of Translation table descriptors hold output address[51:50]. The shareability information of Block and Page descriptors for cacheable locations is determined by: • TCR_EL1.SH0 if the VA is translated using tables pointed to by TTBR0_EL1. • TCR_EL1.SH1 if the VA is translated using tables pointed to by TTBR1_EL1. The minimum value of the TCR_EL1.{T0SZ, T1SZ} fields is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. All calculations of the stage 1 base address are modified for tables of fewer than 8 entries so that the table is aligned to 64 bytes. Bits[5:2] of TTBR0_EL1 or TTBR1_EL1 are used to hold bits[51:48] of the output address in all cases. **Note** As FEAT_LVA must be implemented if TCR_EL1.DS == 1, the minimum value of the TCR_EL1.{T0SZ, T1SZ} fields is 12, as determined by that extension. For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00. **Note** This forces alignment of the ranges used by the TLBI range instructions.

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

The Effective value of this bit is 0b0.

Access to this field is **RES0**.

TCMA1, bit [58]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL1, and at EL0 if the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, when address[59:55] = 0b11111.

TCMA1	Meaning
0b0	This control has no effect on the generation of Unchecked accesses at EL1 or EL0.
0b1	All accesses at EL1 and EL0 are Unchecked.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL1, and at EL0 if the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, when address[59:55] = 0b00000.

TCMA0	Meaning
0b0	This control has no effect on the generation of Unchecked accesses at EL1 or EL0.
0b1	All accesses at EL1 and EL0 are Unchecked.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD1, bit [56]

When FEAT_E0PD is implemented:

Faulting control for unprivileged access to any address translated by TTBR1_EL1.

E0PD1	Meaning
0b0	Unprivileged access to any address translated by TTBR1_EL1 will not generate a fault by this mechanism.
0b1	Unprivileged access to any address translated by TTBR1_EL1 will generate a level 0 Translation fault.

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD0, bit [55]

When FEAT_E0PD is implemented:

Faulting control for unprivileged access to any address translated by TTBR0_EL1.

E0PD0	Meaning
0b0	Unprivileged access to any address translated by TTBR0_EL1 will not generate a fault by this mechanism.
0b1	Unprivileged access to any address translated by TTBR0_EL1 will generate a level 0 Translation fault.

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]
When FEAT_SVE is implemented or FEAT_TME is implemented:

Non-Fault translation timing Disable when using TTBR1_EL1.

Controls how a TLB miss is reported in response to a non-fault unprivileged access for a virtual address that is translated using TTBR1_EL1.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

If FEAT_TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

NFD1	Meaning
0b0	Does not affect the handling of a TLB miss on accesses translated using TTBR1_EL1.
0b1	A TLB miss on a virtual address that is translated using TTBR1_EL1 due to the specified access types causes the access to fail without taking an exception. The amount of time that the failure takes to be handled should not predictively leak whether it was caused by a TLB miss or a Permission fault, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]
When FEAT_SVE is implemented or FEAT_TME is implemented:

Non-Fault translation timing Disable when using TTBR0_EL1.

Controls how a TLB miss is reported in response to a non-fault unprivileged access for a virtual address that is translated using TTBR0_EL1.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

If FEAT_TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

NFD0	Meaning
0b0	Does not affect the handling of a TLB miss on accesses translated using TTBR0_EL1.
0b1	A TLB miss on a virtual address that is translated using TTBR0_EL1 due to the specified access types causes the access to fail without taking an exception. The amount of time that the failure takes to be handled should not predictively leak whether it was caused by a TLB miss or a Permission fault, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging'.

TBID1	Meaning
0b0	TCR_EL1.TBI1 applies to Instruction and Data accesses.
0b1	TCR_EL1.TBI1 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR1_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging'.

TBID0	Meaning
0b0	TCR_EL1.TBI0 applies to Instruction and Data accesses.
0b1	TCR_EL1.TBI0 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU162	Meaning
0b0	For translations using TTBR1_EL1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [49]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU161	Meaning
0b0	For translations using TTBR1_EL1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [48]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU160	Meaning
0b0	For translations using TTBR1_EL1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [47]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU159	Meaning
0b0	For translations using TTBR1_EL1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [46]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU062	Meaning
0b0	For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [45]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU061	Meaning
0b0	For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU060	Meaning
0b0	For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [43]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU059	Meaning
0b0	For translations using TTBR0_EL1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [42]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR1_EL1.

HPD1	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL1.

HPD0	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL0 and EL1.

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled.

When the Effective value of TCR_EL1.HA is 0, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL0 and EL1.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Top Byte ignored. Indicates whether the top byte of an address is used for address match for the TTBR1_EL1 region, or ignored and used for tagged addresses.

TBI1	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL1 using AArch64 where the address would be translated by tables pointed to by TTBR1_EL1. It has an effect whether the EL1&0 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL1.TBID1 is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI1 is 1 and bit [55] of the target address to be stored to the PC is 1, then bits[63:56] of that target address are also set to 1 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TBI0, bit [37]

Top Byte ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL1 region, or ignored and used for tagged addresses.

TBI0	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL1 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL1. It has an effect whether the EL1&0 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL1.TBID0 is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI0 is 1 and bit [55] of the target address to be stored to the PC is 0, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AS, bit [36]

ASID Size.

AS	Meaning
0b0	8 bit - the upper 8 bits of TTBR0_EL1 and TTBR1_EL1 are ignored by hardware for every purpose except reading back the register, and are treated as if they are all zeros for when used for allocation and matching entries in the TLB.
0b1	16 bit - the upper 16 bits of TTBR0_EL1 and TTBR1_EL1 are used for allocation and matching in the TLB.

If the implementation has only 8 bits of ASID, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [35]

Reserved, RES0.

IPS, bits [34:32]

Intermediate Physical Address Size.

IPS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.
0b111	56 bits, 64PB.

The value 0b110 represents the following output address sizes:

- For the 64KB translation granule size, if FEAT_LPA is implemented, the value 0b110 represents 52 bits.
- For the 4KB and 16KB translation granule sizes, if FEAT_LPA is implemented and the Effective value of TCR_EL1.DS is 0b1, then the value 0b110 represents 52 bits.
- Otherwise, the value 0b110 behaves as the 0b101 value and represents 48 bits.

If the value of ID_AA64MMFR0_EL1.PARange is 0b0111, and the value of this field is not 0b111 or a value treated as 0b111, then bits[55:52] of every translation table base address are 0b0000 for the stage of translation controlled by TCR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG1, bits [31:30]

Granule size for the TTBR1_EL1.

TG1	Meaning
0b01	16KB.
0b10	4KB.
0b11	64KB.

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using TTBR1_EL1.

SH1	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using TTBR1_EL1.

ORGN1	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using TTBR1_EL1.

IRGN1	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Translation table walk disable for translations using TTBR1_EL1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1_EL1. The encoding of this bit is:

EPD1	Meaning
0b0	Perform translation table walks using TTBR1_EL1.
0b1	A TLB miss on an address that is translated using TTBR1_EL1 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A1, bit [22]

Selects whether TTBR0_EL1 or TTBR1_EL1 defines the ASID. The encoding of this bit is:

A1	Meaning
0b0	TTBR0_EL1.ASID defines the ASID.
0b1	TTBR1_EL1.ASID defines the ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T1SZ, bits [21:16]

The size offset of the memory region addressed by TTBR1_EL1. The region size is $2^{(64-T1SZ)}$ bytes.

The maximum and minimum possible values for T1SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL1.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL1.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL1.

TG0	Meaning
0b00	4KB
0b01	64KB
0b10	16KB

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL1.

SH0	Meaning
0b00	Non-shareable
0b10	Outer Shareable
0b11	Inner Shareable

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL1.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL1.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Translation table walk disable for translations using TTBR0_EL1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR0_EL1. The encoding of this bit is:

EPD0	Meaning
0b0	Perform translation table walks using TTBR0_EL1.
0b1	A TLB miss on an address that is translated using TTBR0_EL1 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL1. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL1.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL1.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TCR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TCR_EL1 or TCR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to TCR_EL1 are masked by TCRMASK_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x120];
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR_EL2;
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR_EL1;
```

```
MSR TCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x120] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL1 = (X[t, 64] AND NOT EffectiveTCRMASK_EL1()) OR (TCR_EL1 AND
EffectiveTCRMASK_EL1());
        else
            TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL2 = (X[t, 64] AND NOT EffectiveTCRMASK_EL2()) OR (TCR_EL2 AND
EffectiveTCRMASK_EL2());
        else
            TCR_EL2 = X[t, 64];
    else
        TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, TCR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x120];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR TCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x120] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_SRMASK is implemented

```
MRS <Xt>, TCRALIAS_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nTCRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x120];
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR_EL2;
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR_EL1;
```

When FEAT_SRMASK is implemented

```
MSR TCRALIAS_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nTCRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x120] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL1 = (X[t, 64] AND NOT EffectiveTCRMASK_EL1()) OR (TCR_EL1 AND
EffectiveTCRMASK_EL1());
        else
            TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL2 = (X[t, 64] AND NOT EffectiveTCRMASK_EL2()) OR (TCR_EL2 AND
EffectiveTCRMASK_EL2());
        else
            TCR_EL2 = X[t, 64];
    else
        TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCR_EL2, Translation Control Register (EL2)

The TCR_EL2 characteristics are:

Purpose

The control register for stage 1 of the EL2, or EL2&0, translation regime:

- When the Effective value of HCR_EL2.E2H is not 1, this register controls stage 1 of the EL2 translation regime, that supports a single VA range, translated using TTBR0_EL2.
- When the Effective value of HCR_EL2.E2H is 1, this register controls stage 1 of the EL2&0 translation regime, that supports both:
  - A lower VA range, translated using TTBR0_EL2.
  - A higher VA range, translated using TTBR1_EL2.

Configuration

AArch64 System register TCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HTCR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TCR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

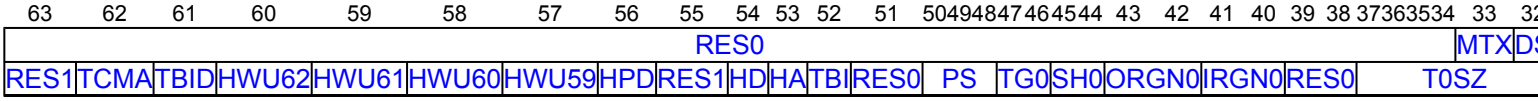
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

TCR_EL2 is a 64-bit register.

Field descriptions

When !ELIsInHost(EL2):



Any of the bits in TCR_EL2 are permitted to be cached in a TLB.

Bits [63:34]

Reserved, RES0.

MTX, bit [33]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL2 is using AArch64 where the data address would be translated by tables pointed to by TTBR0_EL2.

This control has an effect regardless of whether stage 1 of the EL2 translation regime is enabled or not.

MTX	Meaning
0b0	This control has no effect on the PE.
0b1	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as 0b0000 when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [32]
When FEAT_LPA2 is implemented:

This field affects:

- Whether a 52-bit output address can be described by the translation tables of the 4KB or 16KB translation granules.
- The minimum value of TCR_EL2.T0SZ.
- How and where shareability for Block and Page descriptors are encoded.

DS	Meaning
0b0	Bits[49:48] of translation descriptors are RES0. Bits[9:8] in Block and Page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in table descriptors are ignored by hardware. The minimum value of TCR_EL2.T0SZ is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. Output address[51:48] is 0b0000.
0b1	Bits[49:48] of translation descriptors hold output address[49:48]. Bits[9:8] of Translation table descriptors hold output address[51:50]. The shareability information of Block and Page descriptors for cacheable locations is determined by TCR_EL2.SH0. The minimum value of TCR_EL2.T0SZ is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. All calculations of the stage 1 base address are modified for tables of fewer than 8 entries so that the table is aligned to 64 bytes. Bits[5:2] of TTBR0_EL2 are used to hold bits[51:48] of the output address in all cases. **Note** As FEAT_LVA must be implemented if TCR_EL2.DS == 1, the minimum value of the TCR_EL2.T0SZ field is 12, as determined by that extension. For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00. **Note** This forces alignment of the ranges used by the TLBI range instructions.

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

The Effective value of this bit is 0b0.

Access to this field is **RES0**.

Bit [31]

Reserved, RES1.

TCMA, bit [30]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL2 when address [59:56] = 0b0000.

TCMA	Meaning
0b0	This control has no effect on the generation of Unchecked accesses.
0b1	All accesses are Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID, bit [29]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging'.

TBID	Meaning
0b0	TCR_EL2.TBI applies to Instruction and Data accesses.
0b1	TCR_EL2.TBI applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

HWU62	Meaning
0b0	Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

HWU61	Meaning
0b0	Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

HWU60	Meaning
0b0	Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD, bit [24]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL2.

HPD	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled. **Note** In this case, bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be ignored by the PE and are no longer reserved, allowing them to be used by software.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

HD, bit [22]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL2.

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled.

When the Effective value of TCR_EL2.HA is 0, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]
When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL2.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI, bit [20]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL2 region, or ignored and used for tagged addresses.

For more information, see 'Address tagging'.

TBI	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL2 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL2. It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL2.TBID is 1, then this field only applies to Data accesses.

If the value of TBI is 1, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL2.
- An exception taken to EL2.
- An exception return to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

PS, bits [18:16]

Physical Address Size.

PS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.

The value 0b110 represents the following output address sizes:

- For the 64KB translation granule size, if FEAT_LPA is implemented, the value 0b110 represents 52 bits.
- For the 4KB and 16KB translation granule sizes, if FEAT_LPA is implemented and the Effective value of TCR_EL2.DS is 0b1, then the value 0b110 represents 52 bits.
- Otherwise, the value 0b110 behaves as the 0b101 value and represents 48 bits.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL2.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

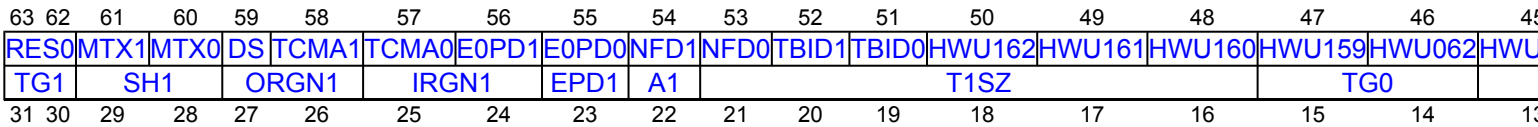
For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ELIsInHost(EL2):



Any of the bits in TCR_EL2, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

Bits [63:62]

Reserved, RES0.

MTX1, bit [61]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL0 and EL2 are using AArch64 where the data address would be translated by tables pointed to by TTBR1_EL2.

This control has an effect regardless of whether stage 1 of the EL2&0 translation regime is enabled or not.

MTX1	Meaning
0b0	This control has no effect on the PE.
0b1	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as 0b1111 when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTX0, bit [60]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL0 and EL2 are using AArch64 where the data address would be translated by tables pointed to by TTBR0_EL2.

This control has an effect regardless of whether stage 1 of the EL2&0 translation regime is enabled or not.

MTX0	Meaning
0b0	This control has no effect on the PE.
0b1	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as 0b0000 when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [59]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or TCR2_EL2.D128 == 0):

This field affects:

- Whether a 52-bit output address can be described by the translation tables of the 4KB or 16KB translation granules.
- The minimum value of TCR_EL2.{T0SZ,T1SZ}.
- How and where shareability for Block and Page descriptors are encoded.

DS	Meaning
0b0	Bits[49:48] of translation descriptors are RES0. Bits[9:8] in Block and Page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in table descriptors are ignored by hardware. The minimum value of the TCR_EL2.{T0SZ, T1SZ} fields is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. Output address[51:48] is 0b0000.
0b1	Bits[49:48] of translation descriptors hold output address[49:48]. Bits[9:8] of Translation table descriptors hold output address[51:50]. The shareability information of Block and Page descriptors for cacheable locations is determined by: • TCR_EL2.SH0 if the VA is an address that is translated using tables pointed to by TTBR0_EL2. • TCR_EL2.SH1 if the VA is an address that is translated using tables pointed to by TTBR1_EL2. The minimum value of the TCR_EL2.{T0SZ, T1SZ} fields is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. All calculations of the stage 1 base address are modified for tables of fewer than 16 entries so that the table is aligned to 64 bytes. Bits[5:2] of TTBR0_EL2 or TTBR1_EL2 are used to hold bits[51:48] of the output address in all cases. **Note** As FEAT_LVA must be implemented if TCR_EL2.DS == 1, the minimum value of the TCR_EL2.{T0SZ, T1SZ} fields is 12, as determined by that extension. For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00. **Note** This forces alignment of the ranges used by the TLBI range instructions.

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0, and the Effective value of this bit is 0b0.

TCMA1, bit [58]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL2, and at EL0 if HCR_EL2.TGE=1, when address[59:55] = 0b11111.

TCMA1	Meaning
0b0	This control has no effect on the generation of Unchecked accesses at EL2 or EL0.
0b1	All accesses are Unchecked.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL2, and at EL0 if HCR_EL2.TGE=1, when address[59:55] = 0b00000.

TCMA0	Meaning
0b0	This control has no effect on the generation of Unchecked accesses at EL2 or EL0.
0b1	All accesses are Unchecked.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD1, bit [56]

When FEAT_E0PD is implemented:

Faulting control for unprivileged access to any address translated by TTBR1_EL2.

E0PD1	Meaning
0b0	Unprivileged access to any address translated by TTBR1_EL2 will not generate a fault by this mechanism.
0b1	Unprivileged access to any address translated by TTBR1_EL2 will generate a level 0 Translation fault.

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD0, bit [55]

When FEAT_E0PD is implemented:

Faulting control for unprivileged access to any address translated by TTBR0_EL2.

E0PD0	Meaning
0b0	Unprivileged access to any address translated by TTBR0_EL2 will not generate a fault by this mechanism.
0b1	Unprivileged access to any address translated by TTBR0_EL2 will generate a level 0 Translation fault.

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]
When FEAT_SVE is implemented or FEAT_TME is implemented:

Non-Fault translation timing Disable when using TTBR1_EL2.

Controls how a TLB miss is reported in response to a non-fault unprivileged access for a virtual address that is translated using TTBR1_EL2.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

If FEAT_TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

NFD1	Meaning
0b0	Does not affect the handling of a TLB miss on accesses translated using TTBR1_EL2.
0b1	A TLB miss on a virtual address that is translated using TTBR1_EL2 due to the specified access types causes the access to fail without taking an exception. The amount of time that the failure takes to be handled should not predictively leak whether it was caused by a TLB miss or a Permission fault, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]
When FEAT_SVE is implemented or FEAT_TME is implemented:

Non-Fault translation timing Disable when using TTBR0_EL2.

Controls how a TLB miss is reported in response to a non-fault unprivileged access for a virtual address that is translated using TTBR0_EL2.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

If FEAT_TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

NFD0	Meaning
0b0	Does not affect the handling of a TLB miss on accesses translated using TTBR0_EL2.
0b1	A TLB miss on a virtual address that is translated using TTBR0_EL2 due to the specified access types causes the access to fail without taking an exception. The amount of time that the failure takes to be handled should not predictively leak whether it was caused by a TLB miss or a Permission fault, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging'.

TBID1	Meaning
0b0	TCR_EL2.TBI1 applies to Instruction and Data accesses.
0b1	TCR_EL2.TBI1 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR1_EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For more information, see 'Address tagging'.

TBID0	Meaning
0b0	TCR_EL2.TBI0 applies to Instruction and Data accesses.
0b1	TCR_EL2.TBI0 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

HWU162	Meaning
0b0	For translations using TTBR1_EL2, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [49]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

HWU161	Meaning
0b0	For translations using TTBR1_EL2, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [48]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

HWU160	Meaning
0b0	For translations using TTBR1_EL2, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [47]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

HWU159	Meaning
0b0	For translations using TTBR1_EL2, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [46]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU062	Meaning
0b0	For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [45]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU061	Meaning
0b0	For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU060	Meaning
0b0	For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [43]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU059	Meaning
0b0	For translations using TTBR0_EL1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [42]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR1_EL2.

HPD1	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL2.

HPD0	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL2.

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled.

When the Effective value of TCR_EL2.HA is 0, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL2.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR1_EL2 region, or ignored and used for tagged addresses.

For more information, see 'Address tagging'.

TBI1	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL2 using AArch64 where the address would be translated by tables pointed to by TTBR1_EL2. It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL2.TBID1 is 1, then this field only applies to Data accesses.

If the value of TBI1 is 1 and bit [55] of the target address to be stored to the PC is 1, then bits[63:56] of that target address are also set to 1 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TBI0, bit [37]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL2 region, or ignored and used for tagged addresses.

For more information, see 'Address tagging'.

TBI0	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL2 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL2. It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL2.TBID0 is 1, then this field only applies to Data accesses.

If the value of TBI0 is 1 and bit [55] of the target address to be stored to the PC is 0, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AS, bit [36]

ASID Size.

AS	Meaning
0b0	8 bit - the upper 8 bits of TTBR0_EL2 and TTBR1_EL2 are ignored by hardware for every purpose except reading back the register, and are treated as if they are all zeros for when used for allocation and matching entries in the TLB.
0b1	16 bit - the upper 16 bits of TTBR0_EL2 and TTBR1_EL2 are used for allocation and matching in the TLB.

If the implementation has only 8 bits of ASID, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [35]

Reserved, RES0.

IPS, bits [34:32]

Intermediate Physical Address Size.

IPS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.
0b111	56 bits, 64PB.

The value 0b110 represents the following output address sizes:

- For the 64KB translation granule size, if FEAT_LPA is implemented, the value 0b110 represents 52 bits.

- For the 4KB and 16KB translation granule sizes, if FEAT_LPA is implemented and the Effective value of TCR_EL2.DS is 0b1, then the value 0b110 represents 52 bits.
- Otherwise, the value 0b110 behaves as the 0b101 value and represents 48 bits.

If the value of ID_AA64MMFR0_EL1.PARange is 0b0111, and the value of this field is not 0b111 or a value treated as 0b111, then bits[55:52] of every translation table base address are 0b0000 for the stage of translation controlled by TCR_EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG1, bits [31:30]

Granule size for the TTBR1_EL2.

TG1	Meaning
0b01	16KB.
0b10	4KB.
0b11	64KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using TTBR1_EL2.

SH1	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using TTBR1_EL2.

ORGN1	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using TTBR1_EL2.

IRGN1	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Translation table walk disable for translations using TTBR1_EL2. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1_EL2. The encoding of this bit is:

EPD1	Meaning
0b0	Perform translation table walks using TTBR1_EL2.
0b1	A TLB miss on an address that is translated using TTBR1_EL2 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A1, bit [22]

Selects whether TTBR0_EL2 or TTBR1_EL2 defines the ASID. The encoding of this bit is:

A1	Meaning
0b0	TTBR0_EL2.ASID defines the ASID.
0b1	TTBR1_EL2.ASID defines the ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T1SZ, bits [21:16]

The size offset of the memory region addressed by TTBR1_EL2. The region size is $2^{(64-T1SZ)}$ bytes.

The maximum and minimum possible values for T1SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL2.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Translation table walk disable for translations using TTBR0_EL2. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR0_EL2. The encoding of this bit is:

EPD0	Meaning
0b0	Perform translation table walks using TTBR0_EL2.
0b1	A TLB miss on an address that is translated using TTBR0_EL2 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TCR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TCR_EL2 or TCR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to TCR_EL2 are masked by TCRMASK_EL2.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR_EL2;
```

```
MSR TCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_SRMASK) then
        TCR_EL2 = (X[t, 64] AND NOT EffectiveTCRMASK_EL2()) OR (TCR_EL2 AND
EffectiveTCRMASK_EL2());
    else
        TCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, TCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x120];
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR_EL2;
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR_EL1;
```

When FEAT_VHE is implemented

```
MSR TCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x120] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL1 = (X[t, 64] AND NOT EffectiveTCRMASK_EL1()) OR (TCR_EL1 AND
EffectiveTCRMASK_EL1());
        else
            TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL2 = (X[t, 64] AND NOT EffectiveTCRMASK_EL2()) OR (TCR_EL2 AND
EffectiveTCRMASK_EL2());
        else
            TCR_EL2 = X[t, 64];
    else
        TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCR_EL3, Translation Control Register (EL3)

The TCR_EL3 characteristics are:

Purpose

The control register for stage 1 of the EL3 translation regime.

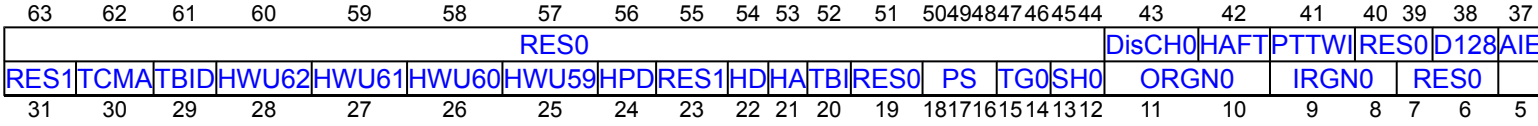
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCR_EL3 are UNDEFINED.

Attributes

TCR_EL3 is a 64-bit register.

Field descriptions



Unless stated otherwise, any of the bits in TCR_EL3 are permitted to be cached in a TLB.

Bits [63:44]

Reserved, RES0.

DisCH0, bit [43]

When FEAT_D128 is implemented and TCR_EL3.D128 == 1:

Disable the Contiguous bit for the Start Table.

DisCH0	Meaning
0b0	The Contiguous bit of Block or Page descriptors of the Start Table is not affected by this field.
0b1	The Contiguous bit of Block or Page descriptors of the Start Table is treated as 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HAFT, bit [42]

When FEAT_HAFT is implemented:

Hardware managed Access Flag for Table descriptors.

Enables the Hardware managed Access Flag for Table descriptors.

HAFT	Meaning
0b0	Hardware managed Access Flag for Table descriptors is disabled.
0b1	Hardware managed Access Flag for Table descriptors is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [41]
When FEAT_THE is implemented:

Permit Translation table walk Incoherence.

Permits RCWS instructions to generate writes that have the Reduced Coherence property.

PTTWI	Meaning
0b0	Write accesses generated by RCWS at EL3 do not have the Reduced Coherence property.
0b1	Write accesses generated by RCWS at EL3 have the Reduced Coherence property.

This bit is permitted to be implemented as a read-only bit with a fixed value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [40:39]

Reserved, RES0.

D128, bit [38]
When FEAT_D128 is implemented:

Enables VMSAv9-128 translation system.

D128	Meaning
0b0	Translation system follows VMSAv8-64 translation process.
0b1	Translation system follows VMSAv9-128 translation process.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIE, bit [37]

When FEAT_AIE is implemented:

Enable Attribute Indexing Extension.

AIE	Meaning
0b0	Attribute Indexing Extension Disabled.
0b1	Attribute Indexing Extension Enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [36]

When FEAT_S1POE is implemented:

Enables Permission Overlay for EL3 accesses.

POE	Meaning
0b0	Permission overlay disabled for EL3 access in stage 1 of EL3 translation regime.
0b1	Permission overlay enabled for EL3 access in stage 1 of EL3 translation regime.

This bit is not permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIE, bit [35]

When FEAT_S1PIE is implemented:

Enables usage of Indirect Permission Scheme.

PIE	Meaning
0b0	Direct permission model.
0b1	Indirect permission model.

This field is RES1 when TCR_EL3.D128 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [34]

When FEAT_THE is implemented:

Protected attribute enable. Enables use of bit[52] of the stage 1 translation table entries as the Protected bit, for translations using TTBR0_EL3.

PnCH	Meaning
0b0	For translations using TTBR0_EL3, bit[52] of each stage 1 translation table entry is not the Protected bit.
0b1	For translations using TTBR0_EL3, bit[52] of each stage 1 translation table entry is the Protected bit.

If bit[52] is used as the Protected bit, it is not used as the Contiguous bit.

This field is RES0 when TCR_EL3.D128 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTX, bit [33]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL3 is using AArch64 where the data address would be translated by tables pointed to by TTBR0_EL3.

This control has an effect regardless of whether stage 1 of the EL3 translation regime is enabled or not.

MTX	Meaning
0b0	This control has no effect on the PE.
0b1	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as 0b0000 when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [32]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or TCR_EL3.D128 == 0):

This field affects:

- Whether a 52-bit output address can be described by the translation tables of the 4KB or 16KB translation granules.
- The minimum value of TCR_EL3.T0SZ.
- How and where shareability for Block and Page descriptors are encoded.

DS	Meaning
0b0	Bits[49:48] of translation descriptors are RES0. Bits[9:8] in Block and Page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in Table descriptors are ignored by hardware. The minimum value of TCR_EL3.T0SZ is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. Output address[51:48] is 0b0000.
0b1	Bits[49:48] of translation descriptors hold output address[49:48]. Bits[9:8] of table translation descriptors hold output address[51:50]. The shareability information of Block and Page descriptors for cacheable locations is determined by TCR_EL3.SH0. The minimum value of TCR_EL3.T0SZ is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. All calculations of the stage 1 base address are modified for tables of fewer than 8 entries so that the table is aligned to 64 bytes. Bits[5:2] of TTBR0_EL3 are used to hold bits[51:48] of the output address in all cases. **Note** As FEAT_LVA must be implemented if TCR_EL3.DS == 1, the minimum value of the TCR_EL3.T0SZ field is 12, as determined by that extension. For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00. **Note** This forces alignment of the ranges used by the TLBI range instructions.

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

The Effective value of this bit is 0b0.

Access to this field is **RES0**.

Bit [31]

Reserved, RES1.

TCMA, bit [30]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL3 when address [59:56] = 0b0000.

TCMA	Meaning
0b0	This control has no effect on the generation of Unchecked accesses.
0b1	All accesses are Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID, bit [29]
When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

TBID	Meaning
0b0	TCR_EL3.TBI applies to Instruction and Data accesses.
0b1	TCR_EL3.TBI applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL3.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

HWU62	Meaning
0b0	Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

HWU61	Meaning
0b0	Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

HWU60	Meaning
0b0	Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD, bit [24]
When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL3.

HPD	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled. **Note** In this case, bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be ignored by the PE, and are no longer reserved, allowing them to be used by software.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

HD, bit [22]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL3.

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled.

When the Effective value of TCR_EL3.HA is 0, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL3.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI, bit [20]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL3 region, or ignored and used for tagged addresses.

TBI	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL3 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL3. It has an effect whether the EL3 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL3.TBID is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI is 1, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL3.
- A exception taken to EL3.
- An exception return to EL3.

For more information, see 'Address tagging'.

Note

This control detrmines the scope of address tagging. It never causes an exception to be generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

PS, bits [18:16]

Physical Address Size.

PS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.
0b111	56 bits, 64PB.

The value 0b110 represents the following output address sizes:

- For the 64KB translation granule size, if FEAT_LPA is implemented, the value 0b110 represents 52 bits.
- For the 4KB and 16KB translation granule sizes, if FEAT_LPA is implemented and the Effective value of TCR_EL3.DS is 0b1, then the value 0b110 represents 52 bits.
- Otherwise, the value 0b110 behaves as the 0b101 value and represents 48 bits.

If the value of ID_AA64MMFR0_EL1.PARange is 0b0111, and the value of this field is not 0b111 or a value treated as 0b111, then bits[55:52] of every translation table base address are 0b0000 for the stage of translation controlled by TCR_EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL3.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL3.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL3.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL3.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL3. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL3.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL3.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR_EL3;
```

```
MSR TCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.TCR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TCR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCRMASK_EL1, Translation Control Masking Register (EL1)

The TCRMASK_EL1 characteristics are:

Purpose

Mask register to prevent updates of fields in TCR_EL1 on writes to TCR_EL1 or TCRALIAS_EL1.

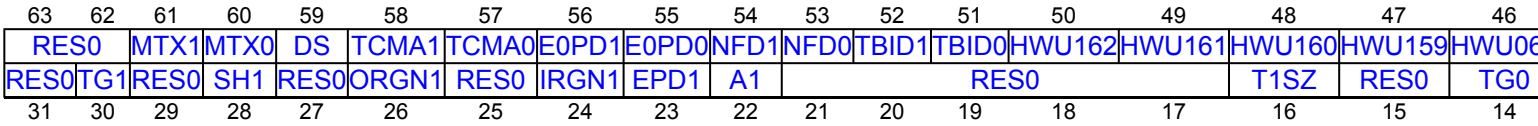
Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCRMASK_EL1 are UNDEFINED.

Attributes

TCRMASK_EL1 is a 64-bit register.

Field descriptions



Bits [63:62]

Reserved, RES0.

MTX1, bit [61]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Mask bit for MTX1.

MTX1	Meaning
0b0	TCR_EL1.MTX1 is writeable.
0b1	TCR_EL1.MTX1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTX0, bit [60]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Mask bit for MTX0.

MTX0	Meaning
0b0	TCR_EL1.MTX0 is writeable.
0b1	TCR_EL1.MTX0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [59]

When FEAT_LPA2 is implemented:

Mask bit for DS.

DS	Meaning
0b0	TCR_EL1.DS is writeable.
0b1	TCR_EL1.DS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA1, bit [58]

When FEAT_MTE2 is implemented:

Mask bit for TCMA1.

TCMA1	Meaning
0b0	TCR_EL1.TCMA1 is writeable.
0b1	TCR_EL1.TCMA1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When FEAT_MTE2 is implemented:

Mask bit for TCMA0.

TCMA0	Meaning
0b0	TCR_EL1.TCMA0 is writeable.
0b1	TCR_EL1.TCMA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD1, bit [56]

When FEAT_E0PD is implemented:

Mask bit for E0PD1.

E0PD1	Meaning
0b0	TCR_EL1.E0PD1 is writeable.
0b1	TCR_EL1.E0PD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD0, bit [55]

When FEAT_E0PD is implemented:

Mask bit for E0PD0.

E0PD0	Meaning
0b0	TCR_EL1.E0PD0 is writeable.
0b1	TCR_EL1.E0PD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]

When FEAT_SVE is implemented or FEAT_TME is implemented:

Mask bit for NFD1.

NFD1	Meaning
0b0	TCR_EL1.NFD1 is writeable.
0b1	TCR_EL1.NFD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]

When FEAT_SVE is implemented or FEAT_TME is implemented:

Mask bit for NFD0.

NFD0	Meaning
0b0	TCR_EL1.NFD0 is writeable.
0b1	TCR_EL1.NFD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When FEAT_PAuth is implemented:

Mask bit for TBID1.

TBID1	Meaning
0b0	TCR_EL1.TBID1 is writeable.
0b1	TCR_EL1.TBID1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When FEAT_PAuth is implemented:

Mask bit for TBID0.

TBID0	Meaning
0b0	TCR_EL1.TBID0 is writeable.
0b1	TCR_EL1.TBID0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When FEAT_HPDS2 is implemented:

Mask bit for HWU162.

HWU162	Meaning
0b0	TCR_EL1.HWU162 is writeable.
0b1	TCR_EL1.HWU162 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [49]

When FEAT_HPDS2 is implemented:

Mask bit for HWU161.

HWU161	Meaning
0b0	TCR_EL1.HWU161 is writeable.
0b1	TCR_EL1.HWU161 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [48]

When FEAT_HPDS2 is implemented:

Mask bit for HWU160.

HWU160	Meaning
0b0	TCR_EL1.HWU160 is writeable.
0b1	TCR_EL1.HWU160 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [47]

When FEAT_HPDS2 is implemented:

Mask bit for HWU159.

HWU159	Meaning
0b0	TCR_EL1.HWU159 is writeable.
0b1	TCR_EL1.HWU159 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [46]

When FEAT_HPDS2 is implemented:

Mask bit for HWU062.

HWU062	Meaning
0b0	TCR_EL1.HWU062 is writeable.
0b1	TCR_EL1.HWU062 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [45]

When FEAT_HPDS2 is implemented:

Mask bit for HWU061.

HWU061	Meaning
0b0	TCR_EL1.HWU061 is writeable.
0b1	TCR_EL1.HWU061 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When FEAT_HPDS2 is implemented:

Mask bit for HWU060.

HWU060	Meaning
0b0	TCR_EL1.HWU060 is writeable.
0b1	TCR_EL1.HWU060 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [43]

When FEAT_HPDS2 is implemented:

Mask bit for HWU059.

HWU059	Meaning
0b0	TCR_EL1.HWU059 is writeable.
0b1	TCR_EL1.HWU059 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [42]

When FEAT_HPDS is implemented:

Mask bit for HPD1.

HPD1	Meaning
0b0	TCR_EL1.HPD1 is writeable.
0b1	TCR_EL1.HPD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When FEAT_HPDS is implemented:

Mask bit for HPD0.

HPD0	Meaning
0b0	TCR_EL1.HPD0 is writeable.
0b1	TCR_EL1.HPD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When FEAT_HAFDBS is implemented:

Mask bit for HD.

HD	Meaning
0b0	TCR_EL1.HD is writeable.
0b1	TCR_EL1.HD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When FEAT_HAFDBS is implemented:

Mask bit for HA.

HA	Meaning
0b0	TCR_EL1.HA is writeable.
0b1	TCR_EL1.HA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Mask bit for TBI1.

TBI1	Meaning
0b0	TCR_EL1.TBI1 is writeable.
0b1	TCR_EL1.TBI1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TBI0, bit [37]

Mask bit for TBI0.

TBI0	Meaning
0b0	TCR_EL1.TBI0 is writeable.
0b1	TCR_EL1.TBI0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AS, bit [36]

Mask bit for AS.

AS	Meaning
0b0	TCR_EL1.AS is writeable.
0b1	TCR_EL1.AS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [35:33]

Reserved, RES0.

IPS, bit [32]

Mask bit for IPS.

IPS	Meaning
0b0	TCR_EL1.IPS is writeable.
0b1	TCR_EL1.IPS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [31]

Reserved, RES0.

TG1, bit [30]

Mask bit for TG1.

TG1	Meaning
0b0	TCR_EL1.TG1 is writeable.
0b1	TCR_EL1.TG1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

SH1, bit [28]

Mask bit for SH1.

SH1	Meaning
0b0	TCR_EL1.SH1 is writeable.
0b1	TCR_EL1.SH1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [27]

Reserved, RES0.

ORGN1, bit [26]

Mask bit for ORGN1.

ORGN1	Meaning
0b0	TCR_EL1.ORGN1 is writeable.
0b1	TCR_EL1.ORGN1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [25]

Reserved, RES0.

IRGN1, bit [24]

Mask bit for IRGN1.

IRGN1	Meaning
0b0	TCR_EL1.IRGN1 is writeable.
0b1	TCR_EL1.IRGN1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Mask bit for EPD1.

EPD1	Meaning
0b0	TCR_EL1.EPD1 is writeable.
0b1	TCR_EL1.EPD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A1, bit [22]

Mask bit for A1.

A1	Meaning
0b0	TCR_EL1.A1 is writeable.
0b1	TCR_EL1.A1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [21:17]

Reserved, RES0.

T1SZ, bit [16]

Mask bit for T1SZ.

T1SZ	Meaning
0b0	TCR_EL1.T1SZ is writeable.
0b1	TCR_EL1.T1SZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

TG0, bit [14]

Mask bit for TG0.

TG0	Meaning
0b0	TCR_EL1.TG0 is writeable.
0b1	TCR_EL1.TG0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

SH0, bit [12]

Mask bit for SH0.

SH0	Meaning
0b0	TCR_EL1.SH0 is writeable.
0b1	TCR_EL1.SH0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

ORGN0, bit [10]

Mask bit for ORGN0.

ORGN0	Meaning
0b0	TCR_EL1.ORGN0 is writeable.
0b1	TCR_EL1.ORGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

IRGN0, bit [8]

Mask bit for IRGN0.

IRGN0	Meaning
0b0	TCR_EL1.IRGN0 is writeable.
0b1	TCR_EL1.IRGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Mask bit for EPD0.

EPD0	Meaning
0b0	TCR_EL1.EPD0 is writeable.
0b1	TCR_EL1.EPD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [6:1]

Reserved, RES0.

T0SZ, bit [0]

Mask bit for T0SZ.

T0SZ	Meaning
0b0	TCR_EL1.T0SZ is writeable.
0b1	TCR_EL1.T0SZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing TCRMASK_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TCRMASK_EL1 or TCRMASK_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCRMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nTCRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x330];
    else
        X[t, 64] = TCRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCRMASK_EL2;
    else
        X[t, 64] = TCRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCRMASK_EL1;
```

```
MSR TCRMASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nTCRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x330] = X[t, 64];
    elsif !IsZero(EffectiveTCRMASK_EL1()) then
        UNDEFINED;
    else
        TCRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveTCRMASK_EL2()) then
            UNDEFINED;
        else
            TCRMASK_EL2 = X[t, 64];
    else
        TCRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCRMASK_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, TCRMASK_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x330];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TCRMASK_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCRMASK_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR TCRMASK_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x330] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TCRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TCRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCRMASK_EL2, Translation Control Masking Register (EL2)

The TCRMASK_EL2 characteristics are:

Purpose

Mask register to prevent updates of fields in TCR_EL2 on writes.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCRMASK_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

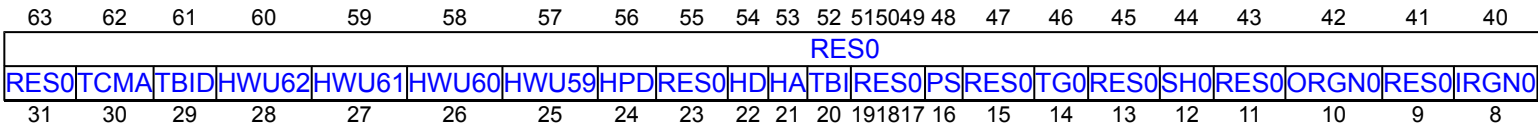
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

TCRMASK_EL2 is a 64-bit register.

Field descriptions

When !ELIsInHost(EL2):



Bits [63:34]

Reserved, RES0.

MTX, bit [33]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Mask bit for MTX.

MTX	Meaning
0b0	TCR_EL2.MTX is writeable.
0b1	TCR_EL2.MTX is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [32]

When FEAT_LPA2 is implemented:

Mask bit for DS.

DS	Meaning
0b0	TCR_EL2.DS is writeable.
0b1	TCR_EL2.DS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES0.

TCMA, bit [30]

When FEAT_MTE2 is implemented:

Mask bit for TCMA.

TCMA	Meaning
0b0	TCR_EL2.TCMA is writeable.
0b1	TCR_EL2.TCMA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID, bit [29]

When FEAT_PAuth is implemented:

Mask bit for TBID.

TBID	Meaning
0b0	TCR_EL2.TBID is writeable.
0b1	TCR_EL2.TBID is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]
When FEAT_HPDS2 is implemented:

Mask bit for HWU62.

HWU62	Meaning
0b0	TCR_EL2.HWU62 is writeable.
0b1	TCR_EL2.HWU62 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]
When FEAT_HPDS2 is implemented:

Mask bit for HWU61.

HWU61	Meaning
0b0	TCR_EL2.HWU61 is writeable.
0b1	TCR_EL2.HWU61 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]
When FEAT_HPDS2 is implemented:

Mask bit for HWU60.

HWU60	Meaning
0b0	TCR_EL2.HWU60 is writeable.
0b1	TCR_EL2.HWU60 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]
When FEAT_HPDS2 is implemented:

Mask bit for HWU59.

HWU59	Meaning
0b0	TCR_EL2.HWU59 is writeable.
0b1	TCR_EL2.HWU59 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD, bit [24]
When FEAT_HPDS is implemented:

Mask bit for HPD.

HPD	Meaning
0b0	TCR_EL2.HPD is writeable.
0b1	TCR_EL2.HPD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES0.

HD, bit [22]
When FEAT_HAFDBS is implemented:

Mask bit for HD.

HD	Meaning
0b0	TCR_EL2.HD is writeable.
0b1	TCR_EL2.HD is not writeable.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When FEAT_HAFDBS is implemented:

Mask bit for HA.

HA	Meaning
0b0	TCR_EL2.HA is writeable.
0b1	TCR_EL2.HA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI, bit [20]

Mask bit for TBI.

TBI	Meaning
0b0	TCR_EL2.TBI is writeable.
0b1	TCR_EL2.TBI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [19:17]

Reserved, RES0.

PS, bit [16]

Mask bit for PS.

PS	Meaning
0b0	TCR_EL2.PS is writeable.
0b1	TCR_EL2.PS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

TG0, bit [14]

Mask bit for TG0.

TG0	Meaning
0b0	TCR_EL2.TG0 is writeable.
0b1	TCR_EL2.TG0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

SH0, bit [12]

Mask bit for SH0.

SH0	Meaning
0b0	TCR_EL2.SH0 is writeable.
0b1	TCR_EL2.SH0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

ORGN0, bit [10]

Mask bit for ORGN0.

ORGN0	Meaning
0b0	TCR_EL2.ORGN0 is writeable.
0b1	TCR_EL2.ORGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

IRGN0, bit [8]

Mask bit for IRGN0.

IRGN0	Meaning
0b0	TCR_EL2.IRGN0 is writeable.
0b1	TCR_EL2.IRGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [7:1]

Reserved, RES0.

T0SZ, bit [0]

Mask bit for T0SZ.

T0SZ	Meaning
0b0	TCR_EL2.T0SZ is writeable.
0b1	TCR_EL2.T0SZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

When ELIsInHost(EL2):

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46
RES0		MTX1	MTX0	DS	TCMA1	TCMA0	E0PD1	E0PD0	NFD1	NFD0	TBID1	TBID0	HWU162	HWU161	HWU160	HWU159	HWU06
RES0	TG1	RES0	SH1	RES0	ORGN1	RES0	IRGN1	EPD1	A1	RES0					T1SZ	RES0	TG0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14

Bits [63:62]

Reserved, RES0.

MTX1, bit [61]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Mask bit for MTX1.

MTX1	Meaning
0b0	TCR_EL2.MTX1 is writeable.
0b1	TCR_EL2.MTX1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTX0, bit [60]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Mask bit for MTX0.

MTX0	Meaning
0b0	TCR_EL2.MTX0 is writeable.
0b1	TCR_EL2.MTX0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [59]

When FEAT_LPA2 is implemented:

Mask bit for DS.

DS	Meaning
0b0	TCR_EL2.DS is writeable.
0b1	TCR_EL2.DS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA1, bit [58]

When FEAT_MTE2 is implemented:

Mask bit for TCMA1.

TCMA1	Meaning
0b0	TCR_EL2.TCMA1 is writeable.
0b1	TCR_EL2.TCMA1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When FEAT_MTE2 is implemented:

Mask bit for TCMA0.

TCMA0	Meaning
0b0	TCR_EL2.TCMA0 is writeable.
0b1	TCR_EL2.TCMA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD1, bit [56]

When FEAT_E0PD is implemented:

Mask bit for E0PD1.

E0PD1	Meaning
0b0	TCR_EL2.E0PD1 is writeable.
0b1	TCR_EL2.E0PD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD0, bit [55]

When FEAT_E0PD is implemented:

Mask bit for E0PD0.

E0PD0	Meaning
0b0	TCR_EL2.E0PD0 is writeable.
0b1	TCR_EL2.E0PD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]

When FEAT_SVE is implemented or FEAT_TME is implemented:

Mask bit for NFD1.

NFD1	Meaning
0b0	TCR_EL2.NFD1 is writeable.
0b1	TCR_EL2.NFD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]

When FEAT_SVE is implemented or FEAT_TME is implemented:

Mask bit for NFD0.

NFD0	Meaning
0b0	TCR_EL2.NFD0 is writeable.
0b1	TCR_EL2.NFD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When FEAT_PAuth is implemented:

Mask bit for TBID1.

TBID1	Meaning
0b0	TCR_EL2.TBID1 is writeable.
0b1	TCR_EL2.TBID1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When FEAT_PAuth is implemented:

Mask bit for TBID0.

TBID0	Meaning
0b0	TCR_EL2.TBID0 is writeable.
0b1	TCR_EL2.TBID0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When FEAT_HPDS2 is implemented:

Mask bit for HWU162.

HWU162	Meaning
0b0	TCR_EL2.HWU162 is writeable.
0b1	TCR_EL2.HWU162 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [49]

When FEAT_HPDS2 is implemented:

Mask bit for HWU161.

HWU161	Meaning
0b0	TCR_EL2.HWU161 is writeable.
0b1	TCR_EL2.HWU161 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [48]

When FEAT_HPDS2 is implemented:

Mask bit for HWU160.

HWU160	Meaning
0b0	TCR_EL2.HWU160 is writeable.
0b1	TCR_EL2.HWU160 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [47]

When FEAT_HPDS2 is implemented:

Mask bit for HWU159.

HWU159	Meaning
0b0	TCR_EL2.HWU159 is writeable.
0b1	TCR_EL2.HWU159 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [46]

When FEAT_HPDS2 is implemented:

Mask bit for HWU062.

HWU062	Meaning
0b0	TCR_EL2.HWU062 is writeable.
0b1	TCR_EL2.HWU062 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [45]

When FEAT_HPDS2 is implemented:

Mask bit for HWU061.

HWU061	Meaning
0b0	TCR_EL2.HWU061 is writeable.
0b1	TCR_EL2.HWU061 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When FEAT_HPDS2 is implemented:

Mask bit for HWU060.

HWU060	Meaning
0b0	TCR_EL2.HWU060 is writeable.
0b1	TCR_EL2.HWU060 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [43]

When FEAT_HPDS2 is implemented:

Mask bit for HWU059.

HWU059	Meaning
0b0	TCR_EL2.HWU059 is writeable.
0b1	TCR_EL2.HWU059 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [42]

When FEAT_HPDS is implemented:

Mask bit for HPD1.

HPD1	Meaning
0b0	TCR_EL2.HPD1 is writeable.
0b1	TCR_EL2.HPD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When FEAT_HPDS is implemented:

Mask bit for HPD0.

HPD0	Meaning
0b0	TCR_EL2.HPD0 is writeable.
0b1	TCR_EL2.HPD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When FEAT_HAFDBS is implemented:

Mask bit for HD.

HD	Meaning
0b0	TCR_EL2.HD is writeable.
0b1	TCR_EL2.HD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When FEAT_HAFDBS is implemented:

Mask bit for HA.

HA	Meaning
0b0	TCR_EL2.HA is writeable.
0b1	TCR_EL2.HA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Mask bit for TBI1.

TBI1	Meaning
0b0	TCR_EL2.TBI1 is writeable.
0b1	TCR_EL2.TBI1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TBI0, bit [37]

Mask bit for TBI0.

TBI0	Meaning
0b0	TCR_EL2.TBI0 is writeable.
0b1	TCR_EL2.TBI0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AS, bit [36]

Mask bit for AS.

AS	Meaning
0b0	TCR_EL2.AS is writeable.
0b1	TCR_EL2.AS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [35:33]

Reserved, RES0.

IPS, bit [32]

Mask bit for IPS.

IPS	Meaning
0b0	TCR_EL2.IPS is writeable.
0b1	TCR_EL2.IPS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [31]

Reserved, RES0.

TG1, bit [30]

Mask bit for TG1.

TG1	Meaning
0b0	TCR_EL2.TG1 is writeable.
0b1	TCR_EL2.TG1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

SH1, bit [28]

Mask bit for SH1.

SH1	Meaning
0b0	TCR_EL2.SH1 is writeable.
0b1	TCR_EL2.SH1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [27]

Reserved, RES0.

ORGN1, bit [26]

Mask bit for ORGN1.

ORGN1	Meaning
0b0	TCR_EL2.ORGN1 is writeable.
0b1	TCR_EL2.ORGN1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [25]

Reserved, RES0.

IRGN1, bit [24]

Mask bit for IRGN1.

IRGN1	Meaning
0b0	TCR_EL2.IRGN1 is writeable.
0b1	TCR_EL2.IRGN1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Mask bit for EPD1.

EPD1	Meaning
0b0	TCR_EL2.EPD1 is writeable.
0b1	TCR_EL2.EPD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A1, bit [22]

Mask bit for A1.

A1	Meaning
0b0	TCR_EL2.A1 is writeable.
0b1	TCR_EL2.A1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [21:17]

Reserved, RES0.

T1SZ, bit [16]

Mask bit for T1SZ.

T1SZ	Meaning
0b0	TCR_EL2.T1SZ is writeable.
0b1	TCR_EL2.T1SZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

TG0, bit [14]

Mask bit for TG0.

TG0	Meaning
0b0	TCR_EL2.TG0 is writeable.
0b1	TCR_EL2.TG0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

SH0, bit [12]

Mask bit for SH0.

SH0	Meaning
0b0	TCR_EL2.SH0 is writeable.
0b1	TCR_EL2.SH0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

ORGN0, bit [10]

Mask bit for ORGN0.

ORGN0	Meaning
0b0	TCR_EL2.ORGN0 is writeable.
0b1	TCR_EL2.ORGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

IRGN0, bit [8]

Mask bit for IRGN0.

IRGN0	Meaning
0b0	TCR_EL2.IRGN0 is writeable.
0b1	TCR_EL2.IRGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Mask bit for EPD0.

EPD0	Meaning
0b0	TCR_EL2.EPD0 is writeable.
0b1	TCR_EL2.EPD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [6:1]

Reserved, RES0.

T0SZ, bit [0]

Mask bit for T0SZ.

T0SZ	Meaning
0b0	TCR_EL2.T0SZ is writeable.
0b1	TCR_EL2.T0SZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing TCRMASK_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TCRMASK_EL2 or TCRMASK_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCRMASK_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TCRMASK_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCRMASK_EL2;
```

```
MSR TCRMASK_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsZero(EffectiveTCRMASK_EL2()) then
        UNDEFINED;
    else
        TCRMASK_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCRMASK_EL2 = X[t, 64];
```

```
MRS <Xt>, TCRMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGRTR2_EL2.nTCRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x330];
    else
        X[t, 64] = TCRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCRMASK_EL2;
    else
        X[t, 64] = TCRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCRMASK_EL1;
```

```
MSR TCRMASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HFGWTR2_EL2.nTCRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x330] = X[t, 64];
    elsif !IsZero(EffectiveTCRMASK_EL1()) then
        UNDEFINED;
    else
        TCRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveTCRMASK_EL2()) then
            UNDEFINED;
        else
            TCRMASK_EL2 = X[t, 64];
    else
        TCRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCRMASK_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TFSR_EL1, Tag Fault Status Register (EL1)

The TFSR_EL1 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL1 that are not taken precisely.

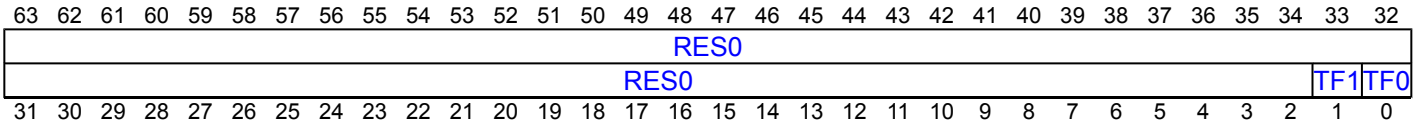
Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to TFSR_EL1 are UNDEFINED.

Attributes

TFSR_EL1 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

TF1, bit [1]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b1` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TF0, bit [0]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b0` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TFSR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TFSR_EL1 or TFSR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TFSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x190];
    else
        X[t, 64] = TFSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TFSR_EL2;
    else
        X[t, 64] = TFSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TFSR_EL1;
```

```
MSR TFSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x190] = X[t, 64];
    else
        TFSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TFSR_EL2 = X[t, 64];
    else
        TFSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TFSR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, TFSR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x190];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TFSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TFSR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR TFSR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x190] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TFSR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

MRS <Xt>, TFSR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif EL2Enabled() && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TFSR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TFSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TFSR_EL2;
```

MSR TFSR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif EL2Enabled() && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TFSR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TFSR_EL2, Tag Fault Status Register (EL2)

The TFSR_EL2 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL2 that are not taken precisely.

Configuration

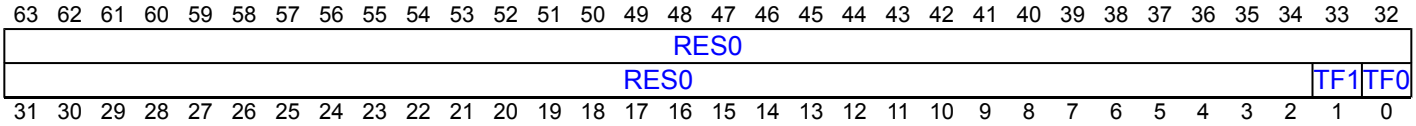
This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to TFSR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

TFSR_EL2 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

TF1, bit [1]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b1` occurs.

When the Effective value of HCR_EL2.E2H is not 1, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TF0, bit [0]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b0` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TFSR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TFSR_EL2 or TFSR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TFSR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif EL2Enabled() && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TFSR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TFSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TFSR_EL2;
```

```
MSR TFSR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif EL2Enabled() && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TFSR_EL2 = X[t, 64];
```

MRS <Xt>, TFSR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x190];
    else
        X[t, 64] = TFSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TFSR_EL2;
    else
        X[t, 64] = TFSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TFSR_EL1;
```

```
MSR TFSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x190] = X[t, 64];
    else
        TFSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TFSR_EL2 = X[t, 64];
    else
        TFSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TFSR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TFSR_EL3, Tag Fault Status Register (EL3)

The TFSR_EL3 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL3 that are not taken precisely.

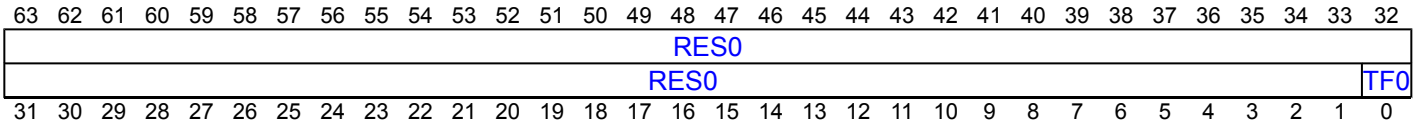
Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to TFSR_EL3 are UNDEFINED.

Attributes

TFSR_EL3 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

TF0, bit [0]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b0` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TFSR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TFSR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TFSR_EL3;
```

```
MSR TFSR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TFSR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TFSRE0_EL1, Tag Fault Status Register (EL0).

The TFSRE0_EL1 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL0 that are not taken precisely.

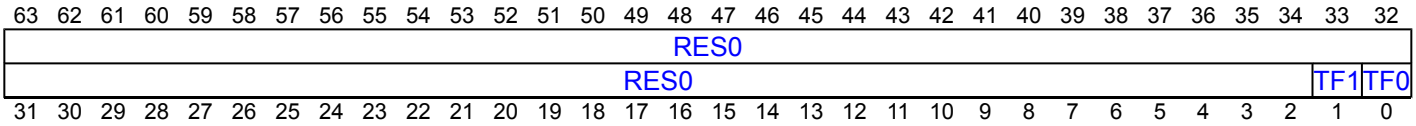
Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to TFSRE0_EL1 are UNDEFINED.

Attributes

TFSRE0_EL1 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

TF1, bit [1]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b1` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TF0, bit [0]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b0` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TFSRE0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TFSRE0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b001

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TFSRE0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TFSRE0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TFSRE0_EL1;
```

```
MSR TFSRE0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b001

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TFSRE0_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE1, TLBI ALLE1NXS, TLB Invalidate All, EL1

The TLBI ALLE1, TLBI ALLE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation only applies to the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ALLE1, TLBI ALLE1NXS are UNDEFINED.

Attributes

TLBI ALLE1, TLBI ALLE1NXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI ALLE1, TLBI ALLE1NXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ALLE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0111	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_AllAttr, X[t,
64]);
```

When FEAT_XS is implemented

```
TLBI ALLE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0111	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_ExcludeXS, X[t,
64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE1IS, TLBI ALLE1ISNXS, TLB Invalidate All, EL1, Inner Shareable

The TLBI ALLE1IS, TLBI ALLE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ALLE1IS, TLBI ALLE1ISNXS are UNDEFINED.

Attributes

TLBI ALLE1IS, TLBI ALLE1ISNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI ALLE1IS, TLBI ALLE1ISNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ALLE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_ISH, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_ISH, TLBI_AllAttr, X[t,
64]);
```

When FEAT_XS is implemented

```
TLBI ALLE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_ISH, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_ISH, TLBI_ExcludeXS, X[t,
64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE1OS, TLBI ALLE1OSNXS, TLB Invalidate All, EL1, Outer Shareable

The TLBI ALLE1OS, TLBI ALLE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ALLE1OS, TLBI ALLE1OSNXS are UNDEFINED.

Attributes

TLBI ALLE1OS, TLBI ALLE1OSNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI ALLE1OS, TLBI ALLE1OSNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ALLE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_OSH, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_OSH, TLBI_AllAttr, X[t,
64]);
```

When FEAT_XS is implemented

```
TLBI ALLE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_OSH, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_OSH, TLBI_ExcludeXS, X[t,
64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE2, TLBI ALLE2NXS, TLB Invalidate All, EL2

The TLBI ALLE2, TLBI ALLE2NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any address using the Secure EL2&0 or EL2 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any address using the Non-secure EL2&0 or EL2 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any address using the Realm EL2&0 or EL2 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any address using the Secure EL2&0 or EL2 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any address using the Non-secure EL2&0 or EL2 translation regime.

The invalidation only applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ALLE2, TLBI ALLE2NXS are UNDEFINED.

Attributes

TLBI ALLE2, TLBI ALLE2NXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI ALLE2, TLBI ALLE2NXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ALLE2{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_NSH, TLBI_AllAttr, X[t,
64]);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_NSH, TLBI_AllAttr, X[t,
64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_NSH, TLBI_AllAttr,
X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_NSH, TLBI_AllAttr, X[t,
64]);
```

When FEAT_XS is implemented

```
TLBI ALLE2NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_NSH, TLBI_ExcludeXS, X[t,
64]);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_NSH, TLBI_ExcludeXS, X[t,
64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_NSH, TLBI_ExcludeXS,
X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_NSH, TLBI_ExcludeXS,
X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE2IS, TLBI ALLE2ISNXS, TLB Invalidate All, EL2, Inner Shareable

The TLBI ALLE2IS, TLBI ALLE2ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any address using the Secure EL2&0 or EL2 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any address using the Non-secure EL2&0 or EL2 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any address using the Realm EL2&0 or EL2 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any address using the Secure EL2&0 or EL2 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any address using the Non-secure EL2&0 or EL2 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ALLE2IS, TLBI ALLE2ISNXS are UNDEFINED.

Attributes

TLBI ALLE2IS, TLBI ALLE2ISNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI ALLE2IS, TLBI ALLE2ISNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ALLE2IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_ISH, TLBI_AllAttr, X[t,
64]);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_ISH, TLBI_AllAttr, X[t,
64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_ISH, TLBI_AllAttr,
X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_ISH, TLBI_AllAttr, X[t,
64]);
```

When FEAT_XS is implemented

```
TLBI ALLE2ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_ISH, TLBI_ExcludeXS, X[t,
64]);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_ISH, TLBI_ExcludeXS, X[t,
64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_ISH, TLBI_ExcludeXS,
X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_ISH, TLBI_ExcludeXS,
X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE2OS, TLBI ALLE2OSNXS, TLB Invalidate All, EL2, Outer Shareable

The TLBI ALLE2OS, TLBI ALLE2OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any address using the Secure EL2&0 or EL2 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any address using the Non-secure EL2&0 or EL2 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any address using the Realm EL2&0 or EL2 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any address using the Secure EL2&0 or EL2 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any address using the Non-secure EL2&0 or EL2 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ALLE2OS, TLBI ALLE2OSNXS are UNDEFINED.

Attributes

TLBI ALLE2OS, TLBI ALLE2OSNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI ALLE2OS, TLBI ALLE2OSNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ALLE2OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_OSH, TLBI_AllAttr, X[t,
64]);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_OSH, TLBI_AllAttr, X[t,
64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_OSH, TLBI_AllAttr,
X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_OSH, TLBI_AllAttr, X[t,
64]);
```

When FEAT_XS is implemented

```
TLBI ALLE2OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_OSH, TLBI_ExcludeXS, X[t,
64]);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_OSH, TLBI_ExcludeXS, X[t,
64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Broadcast_OSH, TLBI_ExcludeXS,
X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_OSH, TLBI_ExcludeXS,
X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE3, TLBI ALLE3NXS, TLB Invalidate All, EL3

The TLBI ALLE3, TLBI ALLE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ALLE3, TLBI ALLE3NXS are UNDEFINED.

Attributes

TLBI ALLE3, TLBI ALLE3NXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI ALLE3, TLBI ALLE3NXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ALLE3{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Broadcast_NSH, TLBI_AllAttr, X[t,
64]);
```

When FEAT_XS is implemented

```
TLBI ALLE3NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Broadcast_NSH, TLBI_ExcludeXS, X[t,
64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE3IS, TLBI ALLE3ISNXS, TLB Invalidate All, EL3, Inner Shareable

The TLBI ALLE3IS, TLBI ALLE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ALLE3IS, TLBI ALLE3ISNXS are UNDEFINED.

Attributes

TLBI ALLE3IS, TLBI ALLE3ISNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI ALLE3IS, TLBI ALLE3ISNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ALLE3IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Broadcast_ISH, TLBI_AllAttr, X[t,
64]);
```

When FEAT_XS is implemented

```
TLBI ALLE3ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Broadcast_ISH, TLBI_ExcludeXS, X[t,
64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE3OS, TLBI ALLE3OSNXS, TLB Invalidate All, EL3, Outer Shareable

The TLBI ALLE3OS, TLBI ALLE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ALLE3OS, TLBI ALLE3OSNXS are UNDEFINED.

Attributes

TLBI ALLE3OS, TLBI ALLE3OSNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI ALLE3OS, TLBI ALLE3OSNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ALLE3OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Broadcast_OSH, TLBI_AllAttr, X[t,
64]);
```

When FEAT_XS is implemented

```
TLBI ALLE3OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Broadcast_OSH, TLBI_ExcludeXS, X[t,
64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ASIDE1, TLBI ASIDE1NXS, TLB Invalidate by ASID, EL1

The TLBI ASIDE1, TLBI ASIDE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate an address using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate an address using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

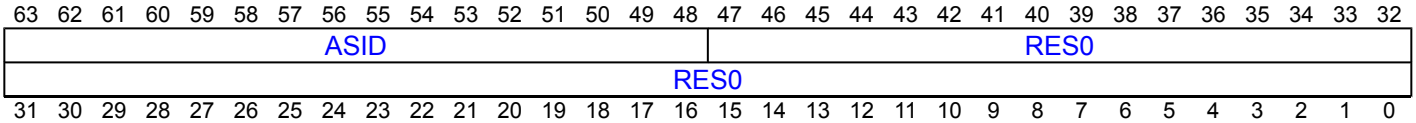
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ASIDE1, TLBI ASIDE1NXS are UNDEFINED.

Attributes

TLBI ASIDE1, TLBI ASIDE1NXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Bits [47:0]

Reserved, RES0.

Executing TLBI ASIDE1, TLBI ASIDE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ASIDE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0111	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIASIDE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI ASIDE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0111	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIASIDE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ASIDE1IS, TLBI ASIDE1ISNXS, TLB Invalidate by ASID, EL1, Inner Shareable

The TLBI ASIDE1IS, TLBI ASIDE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate an address using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate an address using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

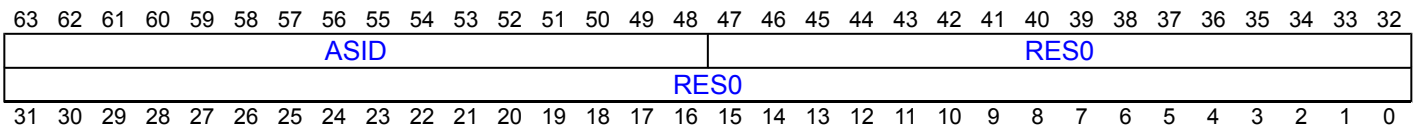
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ASIDE1IS, TLBI ASIDE1ISNXS are UNDEFINED.

Attributes

TLBI ASIDE1IS, TLBI ASIDE1ISNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Bits [47:0]

Reserved, RES0.

Executing TLBI ASIDE1IS, TLBI ASIDE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ASIDE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIASIDE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI ASIDE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2

0b01	0b000	0b1001	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIASIDE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ASIDE1OS, TLBI ASIDE1OSNXS, TLB Invalidate by ASID, EL1, Outer Shareable

The TLBI ASIDE1OS, TLBI ASIDE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate an address using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate an address using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

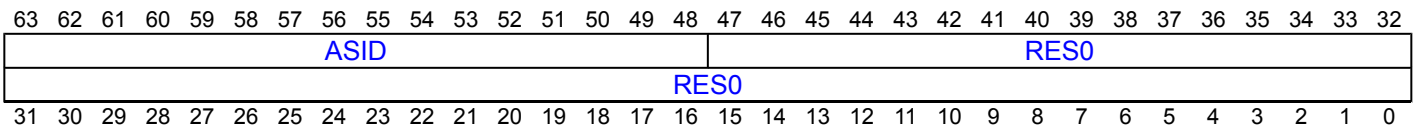
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI ASIDE1OS, TLBI ASIDE1OSNXS are UNDEFINED.

Attributes

TLBI ASIDE1OS, TLBI ASIDE1OSNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Bits [47:0]

Reserved, RES0.

Executing TLBI ASIDE1OS, TLBI ASIDE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI ASIDE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIASIDE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI ASIDE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2

0b01	0b000	0b1001	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIASIDE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2E1, TLBI IPAS2E1NXS, TLB Invalidate by Intermediate Physical Address, Stage 2, EL1

The TLBI IPAS2E1, TLBI IPAS2E1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

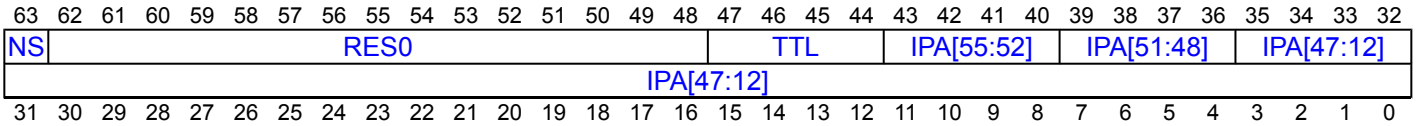
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI IPAS2E1, TLBI IPAS2E1NXS are UNDEFINED.

Attributes

TLBI IPAS2E1, TLBI IPAS2E1NXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

IPA[55:52], bits [43:40]
When FEAT_D128 is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[51:48], bits [39:36]
When FEAT_LPA is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

If ID_AA64MMFR0_EL1.PARange is 0b0111, bits IPA[55:48] form the upper part of the address value.

If ID_AA64MMFR0_EL1.PARange is 0b0110, bits IPA[51:48] form the upper part of the address value and bits IPA[55:52] are RES0.

If ID_AA64MMFR0_EL1.PARange is not 0b0110 and not 0b0111, bits IPA[55:48] are RES0.

Executing TLBI IPAS2E1, TLBI IPAS2E1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI IPAS2E1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI IPAS2E1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS, TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

The TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

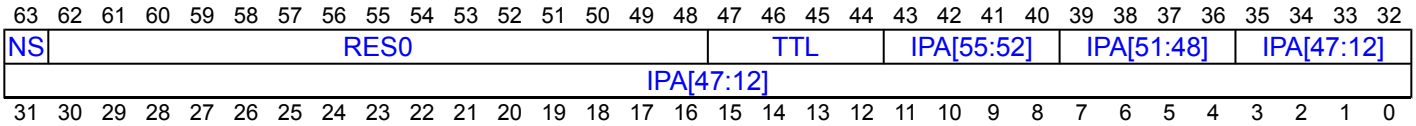
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS are UNDEFINED.

Attributes

TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

IPA[55:52], bits [43:40]
When FEAT_D128 is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[51:48], bits [39:36]
When FEAT_LPA is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

If ID_AA64MMFR0_EL1.PARange is 0b0111, bits IPA[55:48] form the upper part of the address value.

If ID_AA64MMFR0_EL1.PARange is 0b0110, bits IPA[51:48] form the upper part of the address value and bits IPA[55:52] are RES0.

If ID_AA64MMFR0_EL1.PARange is not 0b0110 and not 0b0111, bits IPA[55:48] are RES0.

Executing TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI IPAS2E1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI IPAS2E1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS, TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

The TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

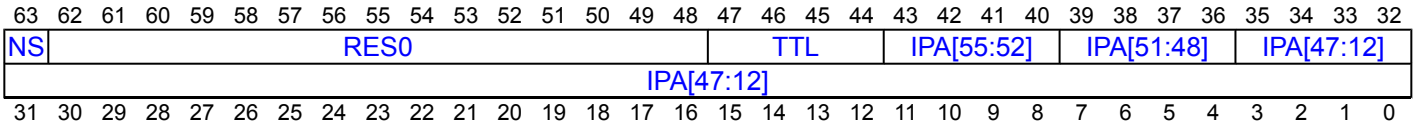
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS are UNDEFINED.

Attributes

TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

IPA[55:52], bits [43:40]
When FEAT_D128 is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[51:48], bits [39:36]

Extension to IPA[47:12]. For more information, see IPA[47:12].

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

If ID_AA64MMFR0_EL1.PARange is 0b0111, bits IPA[55:48] form the upper part of the address value.

If ID_AA64MMFR0_EL1.PARange is 0b0110, bits IPA[51:48] form the upper part of the address value and bits IPA[55:52] are RES0.

If ID_AA64MMFR0_EL1.PARange is not 0b0110 and not 0b0111, bits IPA[55:48] are RES0.

Executing TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI IPAS2E1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Any,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI IPAS2E1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2LE1, TLBI IPAS2LE1NXS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

The TLBI IPAS2LE1, TLBI IPAS2LE1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from the final level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

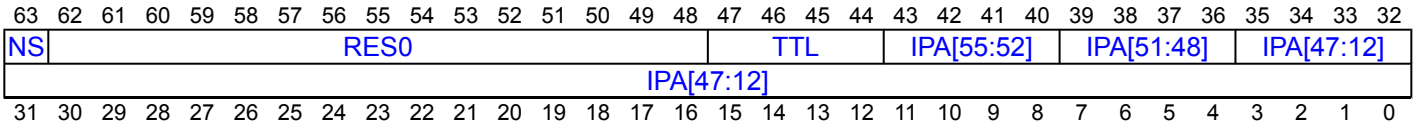
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI IPAS2LE1, TLBI IPAS2LE1NXS are UNDEFINED.

Attributes

TLBI IPAS2LE1, TLBI IPAS2LE1NXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

IPA[55:52], bits [43:40]

When FEAT_D128 is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[51:48], bits [39:36]

When FEAT_LPA is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

If ID_AA64MMFR0_EL1.PARange is 0b0111, bits IPA[55:48] form the upper part of the address value.

If ID_AA64MMFR0_EL1.PARange is 0b0110, bits IPA[51:48] form the upper part of the address value and bits IPA[55:52] are RES0.

If ID_AA64MMFR0_EL1.PARange is not 0b0110 and not 0b0111, bits IPA[55:48] are RES0.

Executing TLBI IPAS2LE1, TLBI IPAS2LE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI IPAS2LE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Last,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI IPAS2LE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Last,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

The TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from the final level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

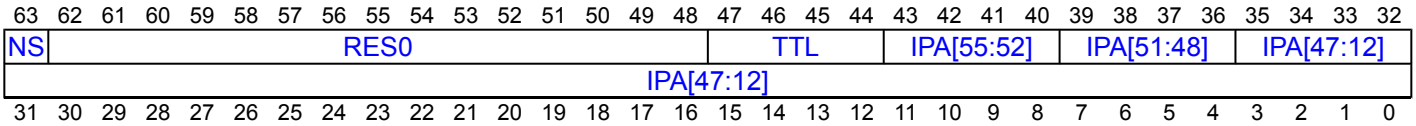
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS are UNDEFINED.

Attributes

TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

IPA[55:52], bits [43:40]
When FEAT_D128 is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[51:48], bits [39:36]
When FEAT_LPA is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

If ID_AA64MMFR0_EL1.PARange is 0b0111, bits IPA[55:48] form the upper part of the address value.

If ID_AA64MMFR0_EL1.PARange is 0b0110, bits IPA[51:48] form the upper part of the address value and bits IPA[55:52] are RES0.

If ID_AA64MMFR0_EL1.PARange is not 0b0110 and not 0b0111, bits IPA[55:48] are RES0.

Executing TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI IPAS2LE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Last,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI IPAS2LE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Last,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

The TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from the final level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

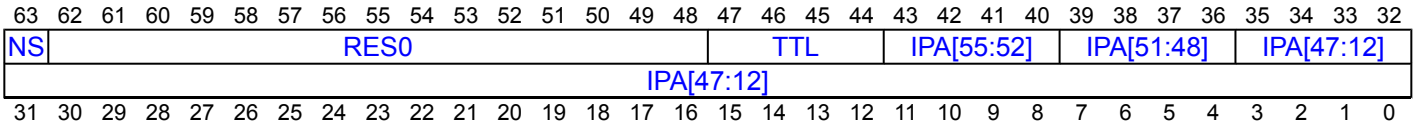
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS are UNDEFINED.

Attributes

TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

IPA[55:52], bits [43:40]
When FEAT_D128 is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[51:48], bits [39:36]

Extension to IPA[47:12]. For more information, see IPA[47:12].

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

If ID_AA64MMFR0_EL1.PARange is 0b0111, bits IPA[55:48] form the upper part of the address value.

If ID_AA64MMFR0_EL1.PARange is 0b0110, bits IPA[51:48] form the upper part of the address value and bits IPA[55:52] are RES0.

If ID_AA64MMFR0_EL1.PARange is not 0b0110 and not 0b0111, bits IPA[55:48] are RES0.

Executing TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI IPAS2LE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b100

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Last,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI IPAS2LE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b100

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Last,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI PAALL, TLB Invalidate GPT Information by PA, All Entries, Local

The TLBI PAALL characteristics are:

Purpose

Invalidates cached copies of GPT entries from TLBs. Details:

- The invalidation applies to TLB entries containing GPT information that relates to a physical address.
- The invalidation applies to all TLB entries containing GPT information.
- The invalidation affects only the TLBs for the PE executing the operation.

The full set of TLB maintenance instructions that invalidate cached GPT entries is: TLBI PAALL, TLBI PAALLOS, TLBI RPALOS, and TLBI RPAOS.

These instructions have the same ordering, observability, and completion behavior as all other TLBI instructions.

Configuration

This instruction is present only when FEAT_RME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI PAALL are UNDEFINED.

Attributes

TLBI PAALL is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI PAALL

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI PAALL{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0111	0b100

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_PAALL(Broadcast_NSH);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI PAALLOS, TLB Invalidate GPT Information by PA, All Entries, Outer Shareable

The TLBI PAALLOS characteristics are:

Purpose

Invalidates cached copies of GPT entries from TLBs. Details:

- The invalidation applies to TLB entries containing GPT information that relates to a physical address.
- The invalidation applies to all TLB entries containing GPT information.
- The invalidation affects all TLBs in the Outer Shareable domain.

The full set of TLB maintenance instructions that invalidate cached GPT entries is: TLBI PAALL, TLBI PAALLOS, TLBI RPALOS, and TLBI RPAOS.

These instructions have the same ordering, observability, and completion behavior as all other TLBI instructions.

Configuration

This instruction is present only when FEAT_RME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI PAALLOS are UNDEFINED.

Attributes

TLBI PAALLOS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI PAALLOS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI PAALLOS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_PAALL(Broadcast_OSH);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RIPAS2E1, TLBI RIPAS2E1NXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1

The TLBI RIPAS2E1, TLBI RIPAS2E1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

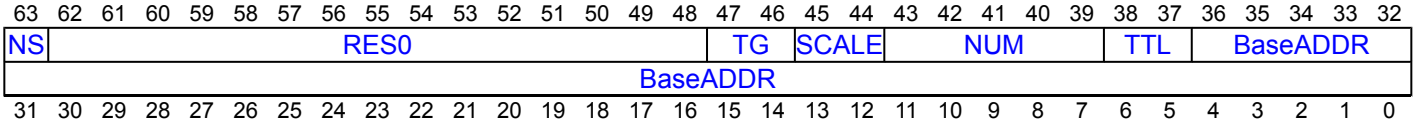
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RIPAS2E1, TLBI RIPAS2E1NXS are UNDEFINED.

Attributes

TLBI RIPAS2E1, TLBI RIPAS2E1NXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and VTCR_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RIPAS2E1, TLBI RIPAS2E1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RIPAS2E1{, <Xt>}
```

op0	op1	CRn	CRm	op2

0b01	0b100	0b1000	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RIPAS2E1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

The TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

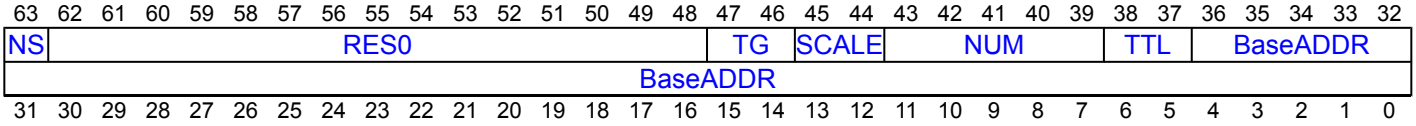
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS are UNDEFINED.

Attributes

TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and VTCR_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RIPAS2E1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2

0b01	0b100	0b1000	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RIPAS2E1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

The TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE+1)}$ * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

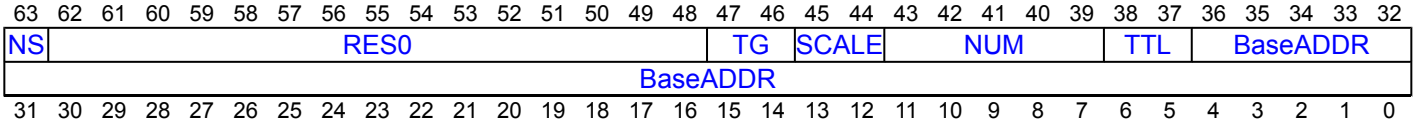
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented, FEAT_TLBIOS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS are UNDEFINED.

Attributes

TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and VTCR_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RIPAS2E1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2

0b01	0b100	0b1000	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Any,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RIPAS2E1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

The TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

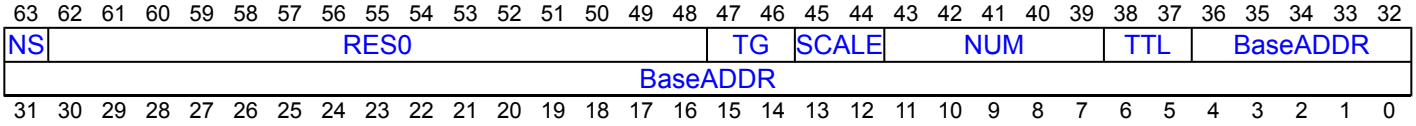
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS are UNDEFINED.

Attributes

TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and VTCR_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RIPAS2LE1{, <Xt>}
```

op0	op1	CRn	CRm	op2

0b01	0b100	0b1000	0b0100	0b110

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RIPAS2LE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b110

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

The TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

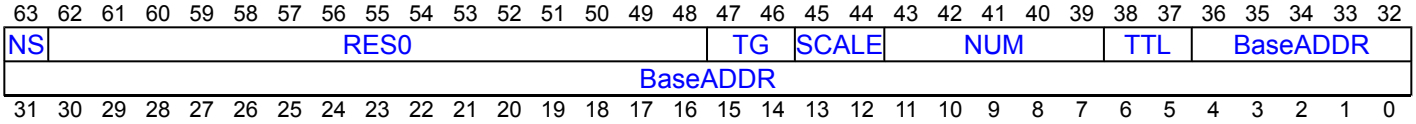
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS are UNDEFINED.

Attributes

TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and VTCR_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RIPAS2LE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RIPAS2LE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

The TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 2 only translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 2 only translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.

- If TTL==10 and BaseADDR[20:12] is not equal to 000000000.

- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

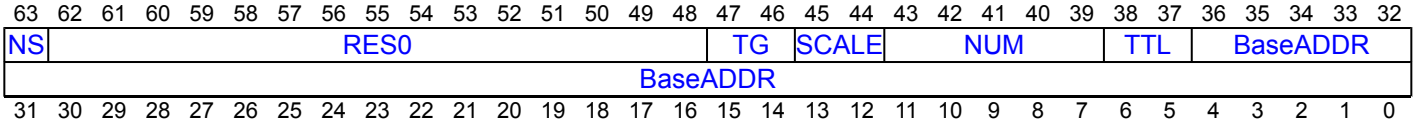
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented, FEAT_TLBIOS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS are UNDEFINED.

Attributes

TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and VTCR_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RIPAS2LE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RIPAS2LE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RPALOS, TLB Range Invalidate GPT Information by PA, Last level, Outer Shareable

The TLBI RPALOS characteristics are:

Purpose

Invalidates cached copies of GPT entries from TLBs. Details:

- The invalidation applies to TLB entries containing GPT information that relates to a physical address.
- The invalidation affects all TLBs in the Outer Shareable domain.
- Invalidates TLB entries containing GPT information from the final level of the GPT walk that relates to the supplied physical address.
- Invalidations are range-based, invalidating TLB entries starting from the address in BaseADDR, within the range as specified by SIZE.

The full set of TLB maintenance instructions that invalidate cached GPT entries is: TLBI PAALL, TLBI PAALLOS, TLBI RPALOS, and TLBI RPAOS.

These instructions have the same ordering, observability, and completion behavior as all other TLBI instructions.

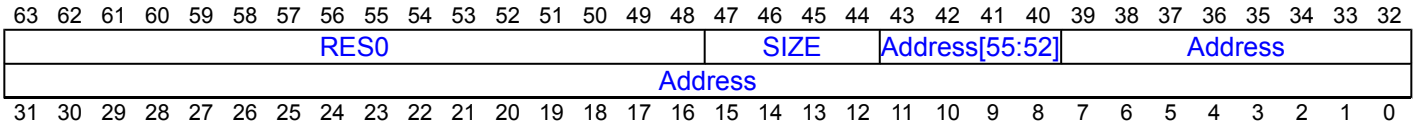
Configuration

This instruction is present only when FEAT_RME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RPALOS are UNDEFINED.

Attributes

TLBI RPALOS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

SIZE, bits [47:44]

Size of the range for invalidation.

If SIZE is a reserved value, no TLB entries are required to be invalidated.

SIZE	Meaning
0b0000	4KB.
0b0001	16KB.
0b0010	64KB.
0b0011	2MB.
0b0100	32MB.
0b0101	512MB.
0b0110	1GB.
0b0111	16GB.
0b1000	64GB.
0b1001	512GB.

All other values are reserved.

If SIZE gives a range smaller than the configured physical granule size in GPCCR_EL3.PGS, then the Effective value of SIZE is taken to be the size configured by GPCCR_EL3.PGS.

If GPCCR_EL3.PGS is configured to a reserved value, no TLB entries are required to be invalidated.

If GPCCR_EL3.PGS is configured to different values at the broadcasting PE and receiving PE, no TLB entries are required to be invalidated at the receiving PE.

Address[55:52], bits [43:40]

When FEAT_D128 is implemented:

Extension to Address. For more information, see Address.

Otherwise:

Reserved, RES0.

Address, bits [39:0]

The starting address for the range of the maintenance instruction.

This field is decoded with reference to the value of GPCCR_EL3.PGS to give BaseADDR as follows:

GPCCR_EL3.PGS	BaseADDR
0b00 (4KB)	BaseADDR[51:12] = Xt[39:0]
0b10 (16KB)	BaseADDR[51:14] = Xt[39:2]
0b01 (64KB)	BaseADDR[51:16] = Xt[39:4]

Other bits of BaseADDR are treated as zero, to give the Effective value of BaseADDR.

If the Effective value of BaseADDR is not aligned to the size of the Effective value of SIZE, no TLB entries are required to be invalidated.

If the Effective value of BaseADDR targets an address above the implemented PA range that ID_AA64MMFR0_EL1.PARange indicates, no TLB entries are required to be invalidated.

If ID_AA64MMFR0_EL1.PARange is 0b0111, Address[55:52] form the upper part of the BaseADDR value. Otherwise, Address[55:52] are RES0.

Executing TLBI RPALOS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RPALOS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RPA(TLBILevel_Last, X[t, 64], Broadcast_OSH);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RPAOS, TLB Range Invalidate GPT Information by PA, Outer Shareable

The TLBI RPAOS characteristics are:

Purpose

Invalidates cached copies of GPT entries from TLBs. Details:

- The invalidation applies to TLB entries containing GPT information that relates to a physical address.
- The invalidation affects all TLBs in the Outer Shareable domain.
- Invalidates TLB entries containing GPT information from all levels of the GPT walk that relates to the supplied physical address.
- Invalidations are range-based, invalidating TLB entries starting from the address in BaseADDR, within the range as specified by SIZE.

The full set of TLB maintenance instructions that invalidate cached GPT entries is: TLBI PAALL, TLBI PAALLOS, TLBI RPALOS, and TLBI RPAOS.

These instructions have the same ordering, observability, and completion behavior as all other TLBI instructions.

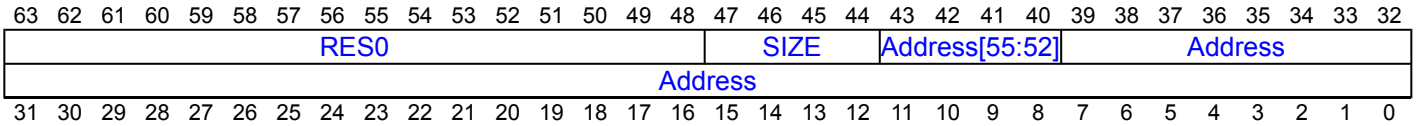
Configuration

This instruction is present only when FEAT_RME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RPAOS are UNDEFINED.

Attributes

TLBI RPAOS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

SIZE, bits [47:44]

Size of the range for invalidation.

If SIZE is a reserved value, no TLB entries are required to be invalidated.

SIZE	Meaning
0b0000	4KB.
0b0001	16KB.
0b0010	64KB.
0b0011	2MB.
0b0100	32MB.
0b0101	512MB.
0b0110	1GB.
0b0111	16GB.
0b1000	64GB.
0b1001	512GB.

All other values are reserved.

If SIZE gives a range smaller than the configured physical granule size in GPCCR_EL3.PGS, then the Effective value of SIZE is taken to be the size configured by GPCCR_EL3.PGS.

If GPCCR_EL3.PGS is configured to a reserved value, no TLB entries are required to be invalidated.

If GPCCR_EL3.PGS is configured to different values at the broadcasting PE and receiving PE, no TLB entries are required to be invalidated at the receiving PE.

Address[55:52], bits [43:40]

When FEAT_D128 is implemented:

Extension to Address. For more information, see Address.

Otherwise:

Reserved, RES0.

Address, bits [39:0]

The starting address for the range of the maintenance instruction.

This field is decoded with reference to the value of GPCCR_EL3.PGS to give BaseADDR as follows:

GPCCR_EL3.PGS	BaseADDR
0b00 (4KB)	BaseADDR[51:12] = Xt[39:0]
0b10 (16KB)	BaseADDR[51:14] = Xt[39:2]
0b01 (64KB)	BaseADDR[51:16] = Xt[39:4]

Other bits of BaseADDR are treated as zero, to give the Effective value of BaseADDR.

If the Effective value of BaseADDR is not aligned to the size of the Effective value of SIZE, no TLB entries are required to be invalidated.

If the Effective value of BaseADDR targets an address above the implemented PA range that ID_AA64MMFR0_EL1.PARange indicates, no TLB entries are required to be invalidated.

If ID_AA64MMFR0_EL1.PARange is 0b0111, Address[55:52] form the upper part of the BaseADDR value. Otherwise, Address[55:52] are RES0.

Executing TLBI RPAOS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RPAOS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RPA(TLBILevel_Any, X[t, 64], Broadcast_OSH);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAAE1, TLBI RVAAE1NXS, TLB Range Invalidate by VA, All ASID, EL1

The TLBI RVAAE1, TLBI RVAAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

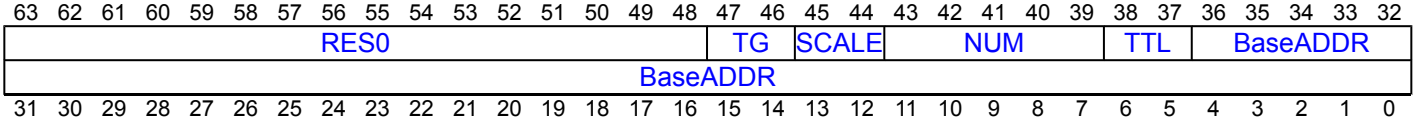
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAAE1, TLBI RVAAE1NXS are UNDEFINED.

Attributes

TLBI RVAAE1, TLBI RVAAE1NXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as `0b0000`.

When using a 16KB translation granule, BaseADDR[15:14] is treated as `0b00`.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAAE1, TLBI RVAAE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAAE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0110	0b011

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAAE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0110	0b011

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAAE1IS, TLBI RVAAE1ISNXS, TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable

The TLBI RVAAE1IS, TLBI RVAAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:
    - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
    - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

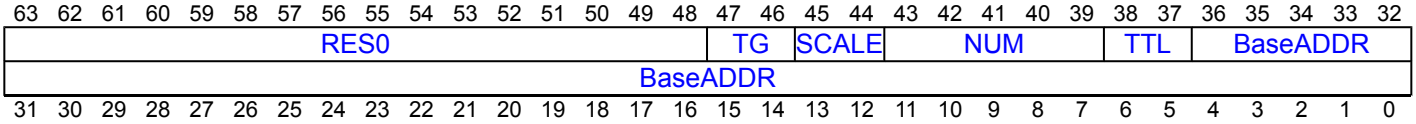
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAAE1IS, TLBI RVAAE1ISNXS are UNDEFINED.

Attributes

TLBI RVAAE1IS, TLBI RVAAE1ISNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAAE1IS, TLBI RVAAE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAAE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAAE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAAE1OS, TLBI RVAAE1OSNXS, TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

The TLBI RVAAE1OS, TLBI RVAAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:
    - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
    - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

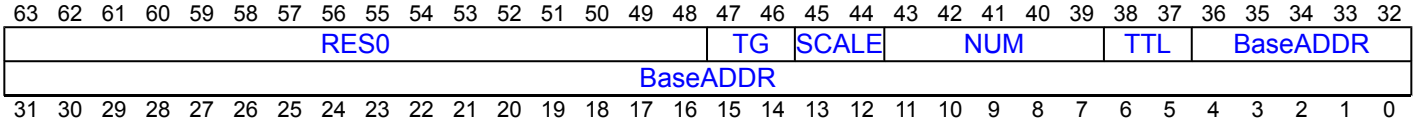
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented, FEAT_TLBIOS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAAE1OS, TLBI RVAAE1OSNXS are UNDEFINED.

Attributes

TLBI RVAAE1OS, TLBI RVAAE1OSNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAAE1OS, TLBI RVAAE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAAE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0101	0b011

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAAE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0101	0b011

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAALE1, TLBI RVAALE1NXS, TLB Range Invalidate by VA, All ASID, Last level, EL1

The TLBI RVAALE1, TLBI RVAALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

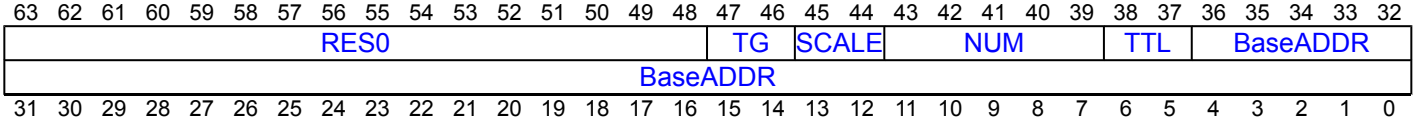
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAALE1, TLBI RVAALE1NXS are UNDEFINED.

Attributes

TLBI RVAALE1, TLBI RVAALE1NXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as `0b0000`.

When using a 16KB translation granule, BaseADDR[15:14] is treated as `0b00`.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAALE1, TLBI RVAALE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAALE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0110	0b111

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAALE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0110	0b111

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAALE1IS, TLBI RVAALE1ISNXS, TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

The TLBI RVAALE1IS, TLBI RVAALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:
    - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
    - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

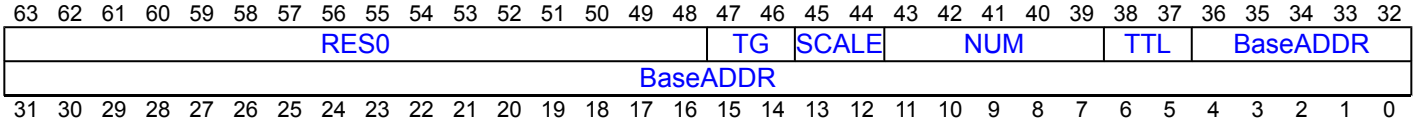
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAALE1IS, TLBI RVAALE1ISNXS are UNDEFINED.

Attributes

TLBI RVAALE1IS, TLBI RVAALE1ISNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAALE1IS, TLBI RVAALE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAALE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0010	0b111

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAALE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0010	0b111

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAALE1OS, TLBI RVAALE1OSNXS, TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

The TLBI RVAALE1OS, TLBI RVAALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

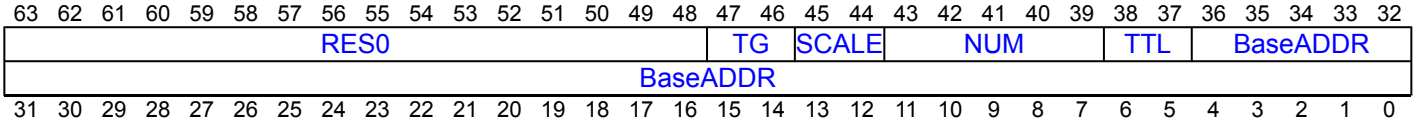
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented, FEAT_TLBIOS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAALE1OS, TLBI RVAALE1OSNXS are UNDEFINED.

Attributes

TLBI RVAALE1OS, TLBI RVAALE1OSNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAALE1OS, TLBI RVAALE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAALE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0101	0b111

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAALE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0101	0b111

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE1, TLBI RVAE1NXS, TLB Range Invalidate by VA, EL1

The TLBI RVAE1, TLBI RVAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE+1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

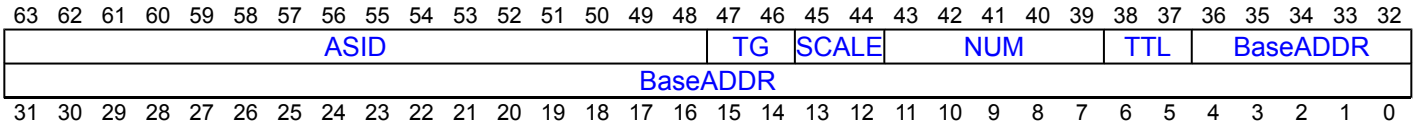
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAE1, TLBI RVAE1NXS are UNDEFINED.

Attributes

TLBI RVAE1, TLBI RVAE1NXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as `0b0000`.

When using a 16KB translation granule, BaseADDR[15:14] is treated as `0b00`.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAE1, TLBI RVAE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE1IS, TLBI RVAE1ISNXS, TLB Range Invalidate by VA, EL1, Inner Shareable

The TLBI RVAE1IS, TLBI RVAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:

    - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:

    - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
    - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

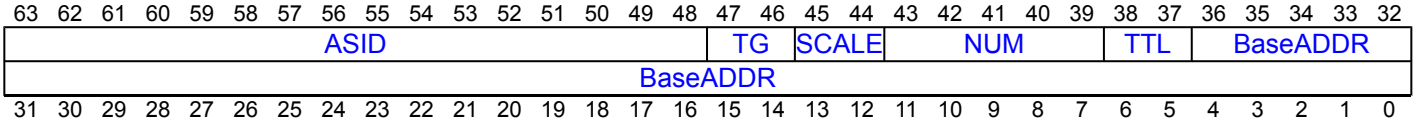
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAE1IS, TLBI RVAE1ISNXS are UNDEFINED.

Attributes

TLBI RVAE1IS, TLBI RVAE1ISNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAE1IS, TLBI RVAE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE1OS, TLBI RVAE1OSNXS, TLB Range Invalidate by VA, EL1, Outer Shareable

The TLBI RVAE1OS, TLBI RVAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:

- If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

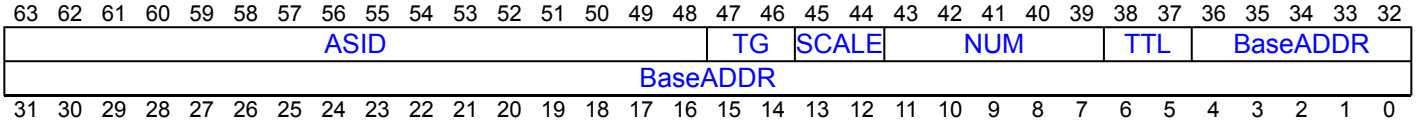
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented, FEAT_TLBIOS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAE1OS, TLBI RVAE1OSNXS are UNDEFINED.

Attributes

TLBI RVAE1OS, TLBI RVAE1OSNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAE1OS, TLBI RVAE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE2, TLBI RVAE2NXS, TLB Range Invalidate by VA, EL2

The TLBI RVAE2, TLBI RVAE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

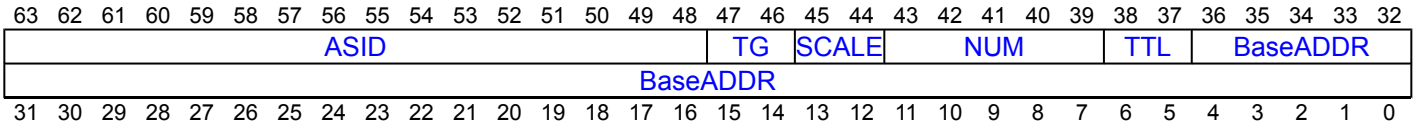
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAE2, TLBI RVAE2NXS are UNDEFINED.

Attributes

TLBI RVAE2, TLBI RVAE2NXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL2.DS == 1) or (FEAT_D128 is implemented and TCR2_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAE2, TLBI RVAE2NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAE2{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAE2NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE2IS, TLBI RVAE2ISNXS, TLB Range Invalidate by VA, EL2, Inner Shareable

The TLBI RVAE2IS, TLBI RVAE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 0000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

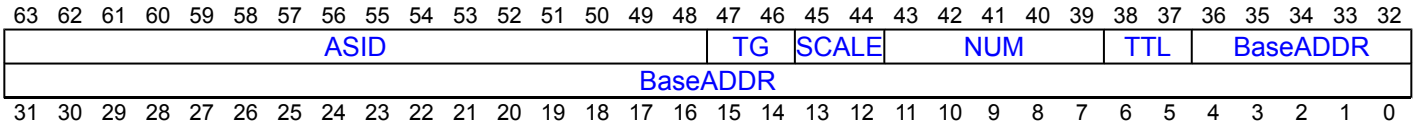
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAE2IS, TLBI RVAE2ISNXS are UNDEFINED.

Attributes

TLBI RVAE2IS, TLBI RVAE2ISNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL2.DS == 1) or (FEAT_D128 is implemented and TCR2_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAE2IS, TLBI RVAE2ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAE2IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAE2ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE2OS, TLBI RVAE2OSNXS, TLB Range Invalidate by VA, EL2, Outer Shareable

The TLBI RVAE2OS, TLBI RVAE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 0000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

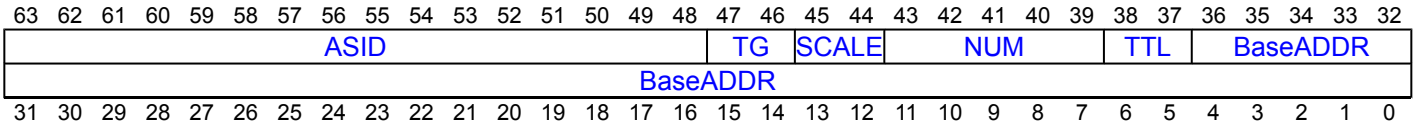
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented, FEAT_TLBIOS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAE2OS, TLBI RVAE2OSNXS are UNDEFINED.

Attributes

TLBI RVAE2OS, TLBI RVAE2OSNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL2.DS == 1) or (FEAT_D128 is implemented and TCR2_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAE2OS, TLBI RVAE2OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAE2OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAE2OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE3, TLBI RVAE3NXS, TLB Range Invalidate by VA, EL3

The TLBI RVAE3, TLBI RVAE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

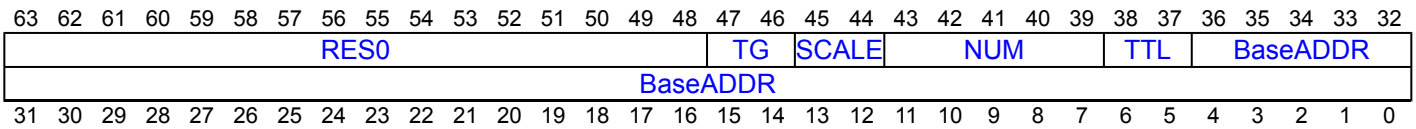
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAE3, TLBI RVAE3NXS are UNDEFINED.

Attributes

TLBI RVAE3, TLBI RVAE3NXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL3.DS == 1) or (FEAT_D128 is implemented and TCR_EL3.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAE3, TLBI RVAE3NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAE3{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAE3NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE3IS, TLBI RVAE3ISNXS, TLB Range Invalidate by VA, EL3, Inner Shareable

The TLBI RVAE3IS, TLBI RVAE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

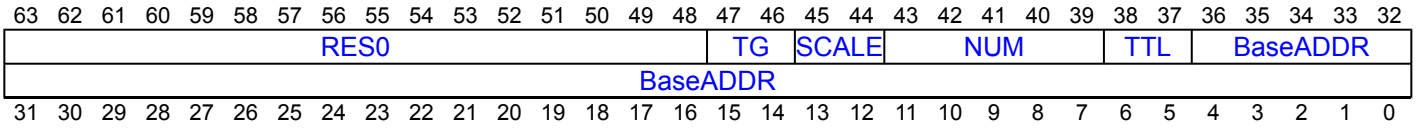
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAE3IS, TLBI RVAE3ISNXS are UNDEFINED.

Attributes

TLBI RVAE3IS, TLBI RVAE3ISNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL3.DS == 1) or (FEAT_D128 is implemented and TCR_EL3.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAE3IS, TLBI RVAE3ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAE3IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAE3ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE3OS, TLBI RVAE3OSNXS, TLB Range Invalidate by VA, EL3, Outer Shareable

The TLBI RVAE3OS, TLBI RVAE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

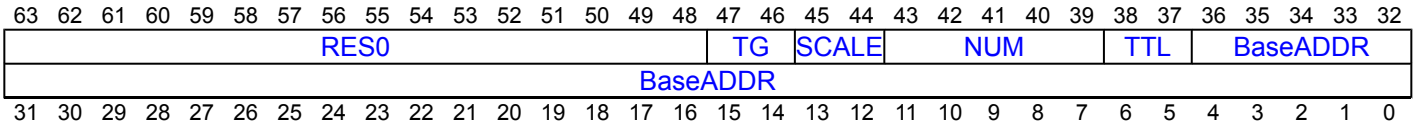
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented, FEAT_TLBIOS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVAE3OS, TLBI RVAE3OSNXS are UNDEFINED.

Attributes

TLBI RVAE3OS, TLBI RVAE3OSNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL3.DS == 1) or (FEAT_D128 is implemented and TCR_EL3.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVAE3OS, TLBI RVAE3OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVAE3OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVAE3OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE1, TLBI RVALE1NXS, TLB Range Invalidate by VA, Last level, EL1

The TLBI RVALE1, TLBI RVALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

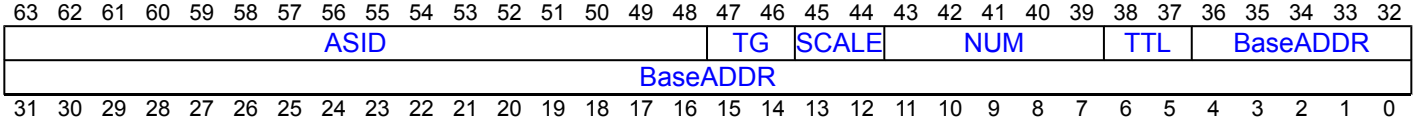
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVALE1, TLBI RVALE1NXS are UNDEFINED.

Attributes

TLBI RVALE1, TLBI RVALE1NXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as `0b0000`.

When using a 16KB translation granule, BaseADDR[15:14] is treated as `0b00`.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVALE1, TLBI RVALE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVALE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVALE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE1IS, TLBI RVALE1ISNXS, TLB Range Invalidate by VA, Last level, EL1, Inner Shareable

The TLBI RVALE1IS, TLBI RVALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:
    - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
    - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

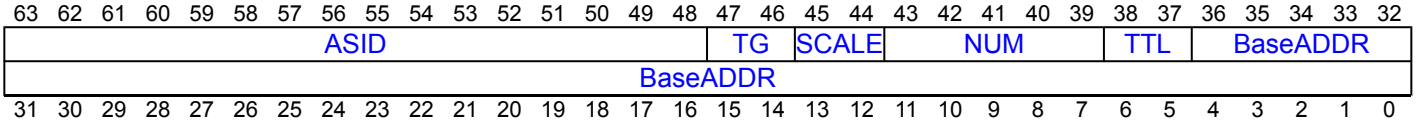
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVALE1IS, TLBI RVALE1ISNXS are UNDEFINED.

Attributes

TLBI RVALE1IS, TLBI RVALE1ISNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVALE1IS, TLBI RVALE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVALE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVALE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE1OS, TLBI RVALE1OSNXS, TLB Range Invalidate by VA, Last level, EL1, Outer Shareable

The TLBI RVALE1OS, TLBI RVALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:
    - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
    - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

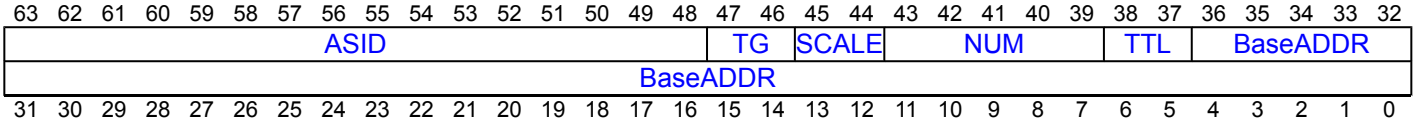
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented, FEAT_TLBIOS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVALE1OS, TLBI RVALE1OSNXS are UNDEFINED.

Attributes

TLBI RVALE1OS, TLBI RVALE1OSNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL1.DS == 1) or (FEAT_D128 is implemented and TCR2_EL1.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVALE1OS, TLBI RVALE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVALE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVALE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE2, TLBI RVALE2NXS, TLB Range Invalidate by VA, Last level, EL2

The TLBI RVALE2, TLBI RVALE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

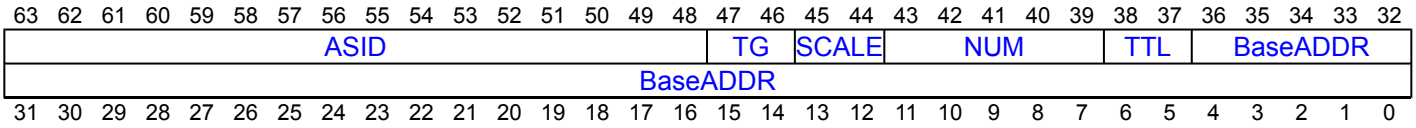
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVALE2, TLBI RVALE2NXS are UNDEFINED.

Attributes

TLBI RVALE2, TLBI RVALE2NXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL2.DS == 1) or (FEAT_D128 is implemented and TCR2_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVALE2, TLBI RVALE2NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVALE2{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVALE2NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE2IS, TLBI RVALE2ISNXS, TLB Range Invalidate by VA, Last level, EL2, Inner Shareable

The TLBI RVALE2IS, TLBI RVALE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

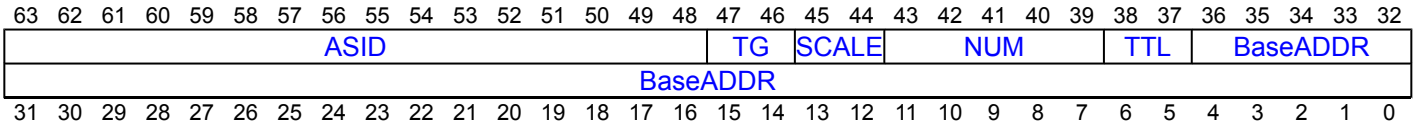
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVALE2IS, TLBI RVALE2ISNXS are UNDEFINED.

Attributes

TLBI RVALE2IS, TLBI RVALE2ISNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL2.DS == 1) or (FEAT_D128 is implemented and TCR2_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVALE2IS, TLBI RVALE2ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVALE2IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVALE2ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE2OS, TLBI RVALE2OSNXS, TLB Range Invalidate by VA, Last level, EL2, Outer Shareable

The TLBI RVALE2OS, TLBI RVALE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

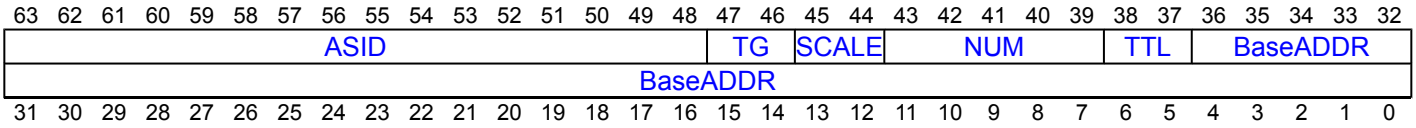
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented, FEAT_TLBIOS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVALE2OS, TLBI RVALE2OSNXS are UNDEFINED.

Attributes

TLBI RVALE2OS, TLBI RVALE2OSNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL2.DS == 1) or (FEAT_D128 is implemented and TCR2_EL2.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVALE2OS, TLBI RVALE2OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVALE2OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVALE2OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE3, TLBI RVALE3NXS, TLB Range Invalidate by VA, Last level, EL3

The TLBI RVALE3, TLBI RVALE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

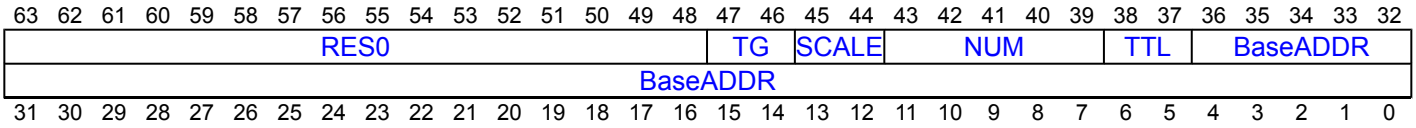
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVALE3, TLBI RVALE3NXS are UNDEFINED.

Attributes

TLBI RVALE3, TLBI RVALE3NXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL3.DS == 1) or (FEAT_D128 is implemented and TCR_EL3.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVALE3, TLBI RVALE3NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVALE3{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVALE3NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE3IS, TLBI RVALE3ISNXS, TLB Range Invalidate by VA, Last level, EL3, Inner Shareable

The TLBI RVALE3IS, TLBI RVALE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

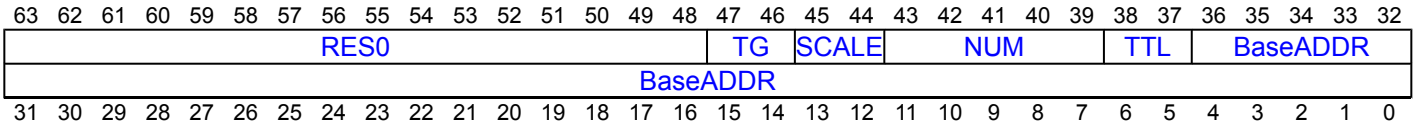
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVALE3IS, TLBI RVALE3ISNXS are UNDEFINED.

Attributes

TLBI RVALE3IS, TLBI RVALE3ISNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL3.DS == 1) or (FEAT_D128 is implemented and TCR_EL3.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVALE3IS, TLBI RVALE3ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVALE3IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVALE3ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE3OS, TLBI RVALE3OSNXS, TLB Range Invalidate by VA, Last level, EL3, Outer Shareable

The TLBI RVALE3OS, TLBI RVALE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk up to the level indicated in the TTL hint.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 64-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

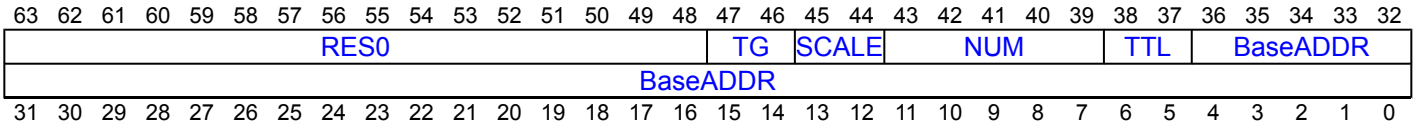
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented, FEAT_TLBIOS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI RVALE3OS, TLBI RVALE3OSNXS are UNDEFINED.

Attributes

TLBI RVALE3OS, TLBI RVALE3OSNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

BaseADDR, bits [36:0]

When (FEAT_LPA2 is implemented and TCR_EL3.DS == 1) or (FEAT_D128 is implemented and TCR_EL3.D128 == 1):

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing TLBI RVALE3OS, TLBI RVALE3OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RVALE3OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI RVALE3OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIRANGE) && IsFeatureImplemented(FEAT_TLBIOS) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAAE1, TLBI VAAE1NXS, TLB Invalidate by VA, All ASID, EL1

The TLBI VAAE1, TLBI VAAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

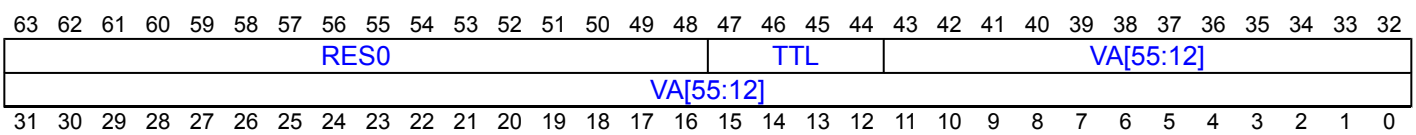
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAAE1, TLBI VAAE1NXS are UNDEFINED.

Attributes

TLBI VAAE1, TLBI VAAE1NXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAAE1, TLBI VAAE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAAE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0111	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAAE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0111	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAAE1IS, TLBI VAAE1ISNXS, TLB Invalidate by VA, All ASID, EL1, Inner Shareable

The TLBI VAAE1IS, TLBI VAAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

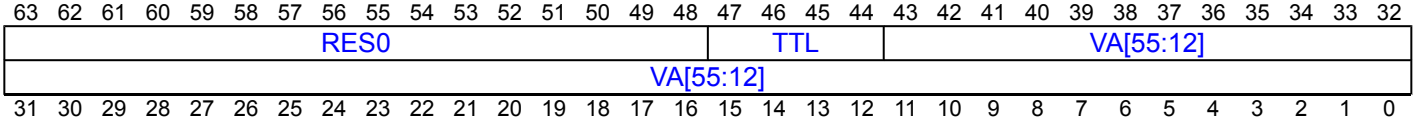
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAAE1IS, TLBI VAAE1ISNXS are UNDEFINED.

Attributes

TLBI VAAE1IS, TLBI VAAE1ISNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAAE1IS, TLBI VAAE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAAE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0011	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAAE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0011	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAAE1OS, TLBI VAAE1OSNXS, TLB Invalidate by VA, All ASID, EL1, Outer Shareable

The TLBI VAAE1OS, TLBI VAAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

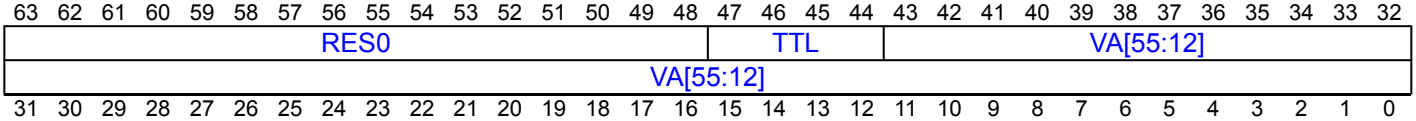
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAAE1OS, TLBI VAAE1OSNXS are UNDEFINED.

Attributes

TLBI VAAE1OS, TLBI VAAE1OSNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAAE1OS, TLBI VAAE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAAE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0001	0b011

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAAE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0001	0b011

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAALE1, TLBI VAALE1NXS, TLB Invalidate by VA, All ASID, Last level, EL1

The TLBI VAALE1, TLBI VAALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

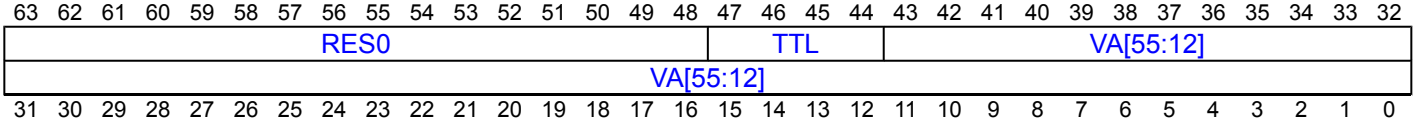
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAALE1, TLBI VAALE1NXS are UNDEFINED.

Attributes

TLBI VAALE1, TLBI VAALE1NXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAALE1, TLBI VAALE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAALE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0111	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAALE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0111	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAALE1IS, TLBI VAALE1ISNXS, TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

The TLBI VAALE1IS, TLBI VAALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

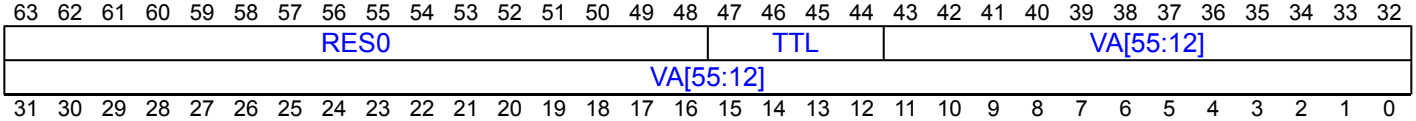
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAALE1IS, TLBI VAALE1ISNXS are UNDEFINED.

Attributes

TLBI VAALE1IS, TLBI VAALE1ISNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAALE1IS, TLBI VAALE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAALE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0011	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAALE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0011	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAALE1OS, TLBI VAALE1OSNXS, TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

The TLBI VAALE1OS, TLBI VAALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

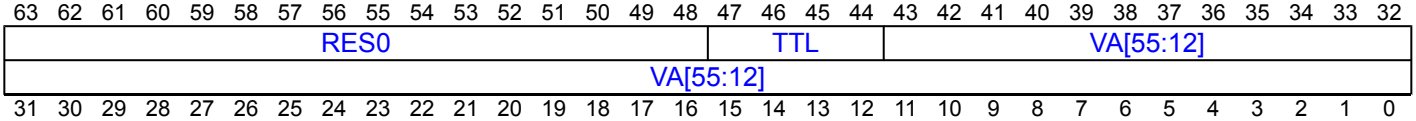
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAALE1OS, TLBI VAALE1OSNXS are UNDEFINED.

Attributes

TLBI VAALE1OS, TLBI VAALE1OSNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAALE1OS, TLBI VAALE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAALE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0001	0b111

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAALE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0001	0b111

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE1, TLBI VAE1NXS, TLB Invalidate by VA, EL1

The TLBI VAE1, TLBI VAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

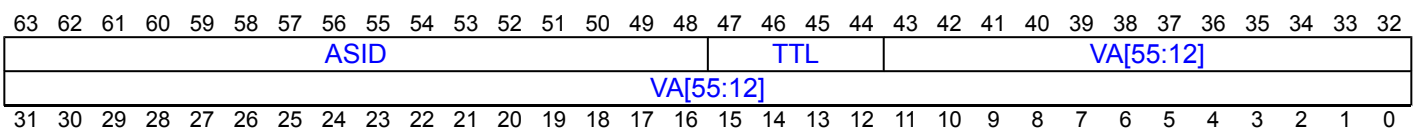
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAE1, TLBI VAE1NXS are UNDEFINED.

Attributes

TLBI VAE1, TLBI VAE1NXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAE1, TLBI VAE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0111	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0111	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE1IS, TLBI VAE1ISNXS, TLB Invalidate by VA, EL1, Inner Shareable

The TLBI VAE1IS, TLBI VAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

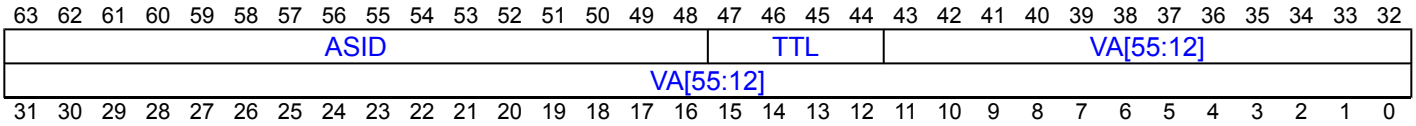
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAE1IS, TLBI VAE1ISNXS are UNDEFINED.

Attributes

TLBI VAE1IS, TLBI VAE1ISNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAE1IS, TLBI VAE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE1OS, TLBI VAE1OSNXS, TLB Invalidate by VA, EL1, Outer Shareable

The TLBI VAE1OS, TLBI VAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

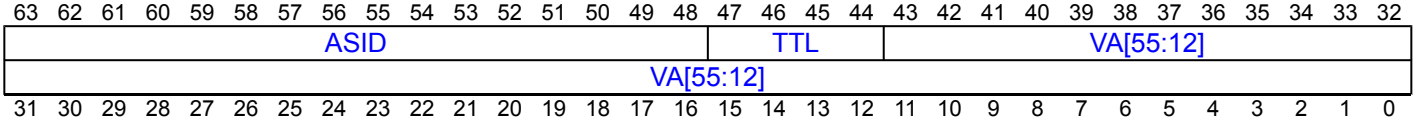
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAE1OS, TLBI VAE1OSNXS are UNDEFINED.

Attributes

TLBI VAE1OS, TLBI VAE1OSNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAE1OS, TLBI VAE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Any,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE2, TLBI VAE2NXS, TLB Invalidate by VA, EL2

The TLBI VAE2, TLBI VAE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

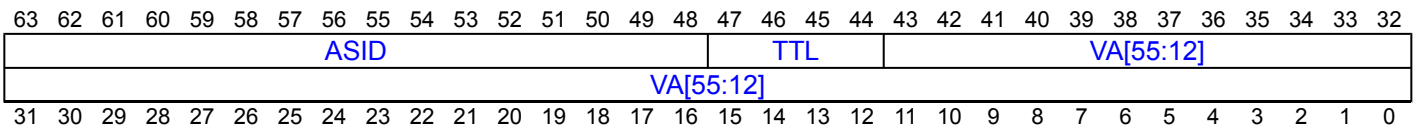
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAE2, TLBI VAE2NXS are UNDEFINED.

Attributes

TLBI VAE2, TLBI VAE2NXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAE2, TLBI VAE2NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAE2{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0111	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAE2NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0111	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE2IS, TLBI VAE2ISNXS, TLB Invalidate by VA, EL2, Inner Shareable

The TLBI VAE2IS, TLBI VAE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

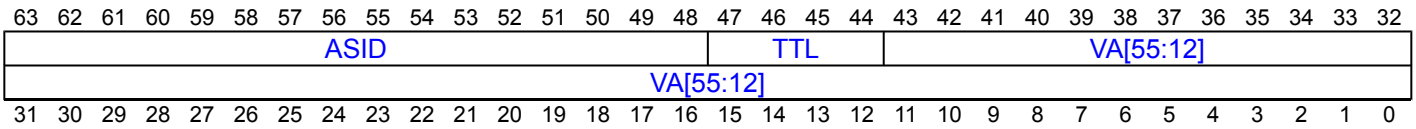
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAE2IS, TLBI VAE2ISNXS are UNDEFINED.

Attributes

TLBI VAE2IS, TLBI VAE2ISNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAE2IS, TLBI VAE2ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAE2IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAE2ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE2OS, TLBI VAE2OSNXS, TLB Invalidate by VA, EL2, Outer Shareable

The TLBI VAE2OS, TLBI VAE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

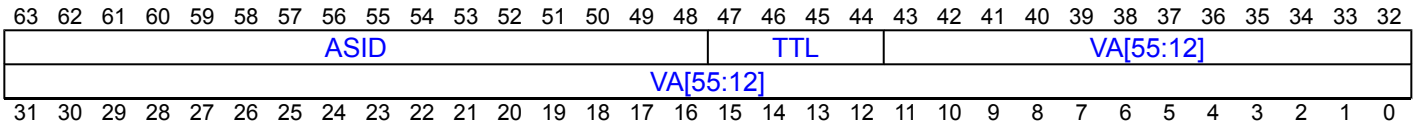
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAE2OS, TLBI VAE2OSNXS are UNDEFINED.

Attributes

TLBI VAE2OS, TLBI VAE2OSNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.

- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAE2OS, TLBI VAE2OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAE2OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAE2OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE3, TLBI VAE3NXS, TLB Invalidate by VA, EL3

The TLBI VAE3, TLBI VAE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

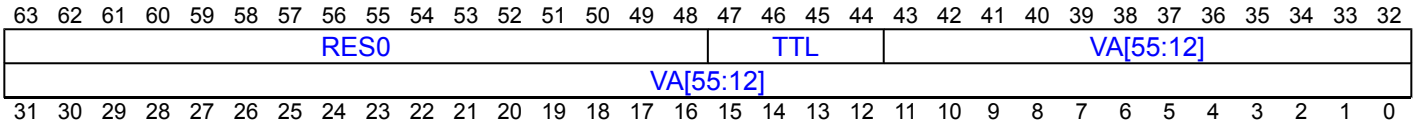
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAE3, TLBI VAE3NXS are UNDEFINED.

Attributes

TLBI VAE3, TLBI VAE3NXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAE3, TLBI VAE3NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAE3{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0111	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAE3NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0111	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE3IS, TLBI VAE3ISNXS, TLB Invalidate by VA, EL3, Inner Shareable

The TLBI VAE3IS, TLBI VAE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

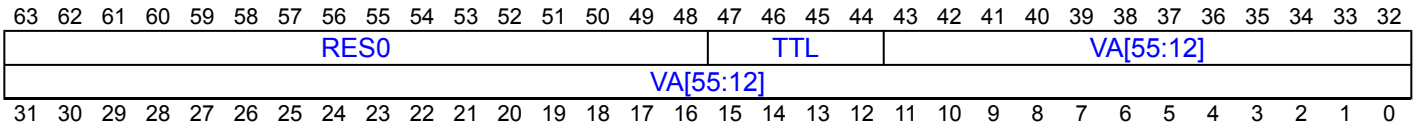
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAE3IS, TLBI VAE3ISNXS are UNDEFINED.

Attributes

TLBI VAE3IS, TLBI VAE3ISNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAE3IS, TLBI VAE3ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAE3IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAE3ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE3OS, TLBI VAE3OSNXS, TLB Invalidate by VA, EL3, Outer Shareable

The TLBI VAE3OS, TLBI VAE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

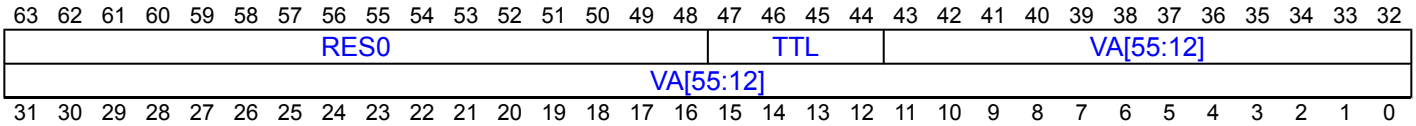
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VAE3OS, TLBI VAE3OSNXS are UNDEFINED.

Attributes

TLBI VAE3OS, TLBI VAE3OSNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VAE3OS, TLBI VAE3OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VAE3OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VAE3OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE1, TLBI VALE1NXS, TLB Invalidate by VA, Last level, EL1

The TLBI VALE1, TLBI VALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

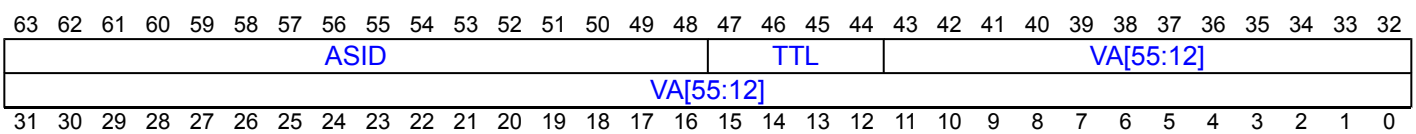
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VALE1, TLBI VALE1NXS are UNDEFINED.

Attributes

TLBI VALE1, TLBI VALE1NXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VALE1, TLBI VALE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VALE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0111	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VALE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0111	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE1IS, TLBI VALE1ISNXS, TLB Invalidate by VA, Last level, EL1, Inner Shareable

The TLBI VALE1IS, TLBI VALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

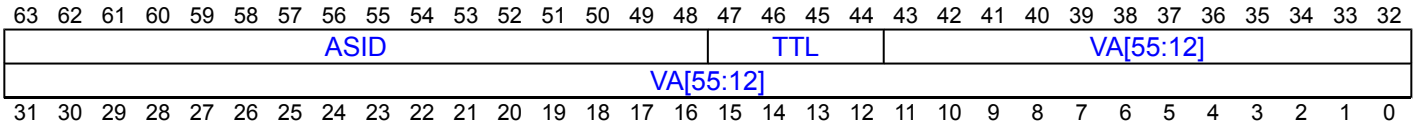
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VALE1IS, TLBI VALE1ISNXS are UNDEFINED.

Attributes

TLBI VALE1IS, TLBI VALE1ISNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VALE1IS, TLBI VALE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VALE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0011	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VALE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0011	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE1OS, TLBI VALE1OSNXS, TLB Invalidate by VA, Last level, EL1, Outer Shareable

The TLBI VALE1OS, TLBI VALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

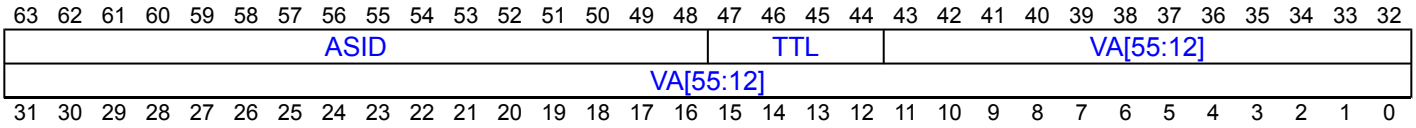
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VALE1OS, TLBI VALE1OSNXS are UNDEFINED.

Attributes

TLBI VALE1OS, TLBI VALE1OSNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VALE1OS, TLBI VALE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VALE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VALE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE2, TLBI VALE2NXS, TLB Invalidate by VA, Last level, EL2

The TLBI VALE2, TLBI VALE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

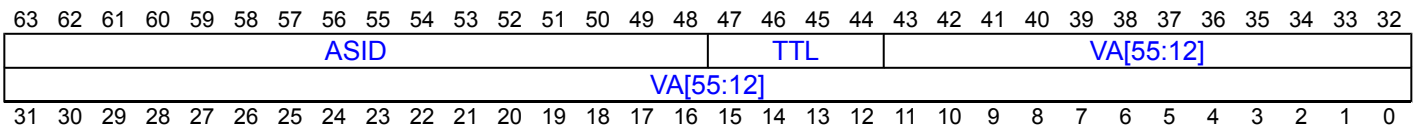
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VALE2, TLBI VALE2NXS are UNDEFINED.

Attributes

TLBI VALE2, TLBI VALE2NXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VALE2, TLBI VALE2NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VALE2{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0111	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VALE2NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0111	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE2IS, TLBI VALE2ISNXS, TLB Invalidate by VA, Last level, EL2, Inner Shareable

The TLBI VALE2IS, TLBI VALE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

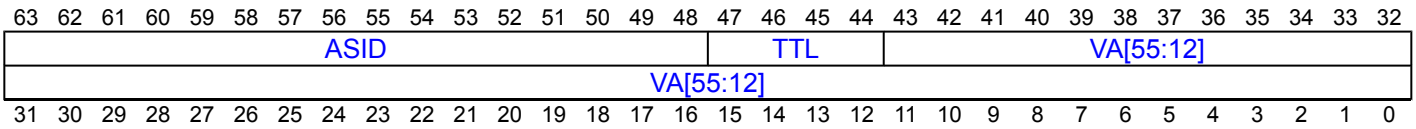
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VALE2IS, TLBI VALE2ISNXS are UNDEFINED.

Attributes

TLBI VALE2IS, TLBI VALE2ISNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VALE2IS, TLBI VALE2ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VALE2IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0011	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VALE2ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0011	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE2OS, TLBI VALE2OSNXS, TLB Invalidate by VA, Last level, EL2, Outer Shareable

The TLBI VALE2OS, TLBI VALE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

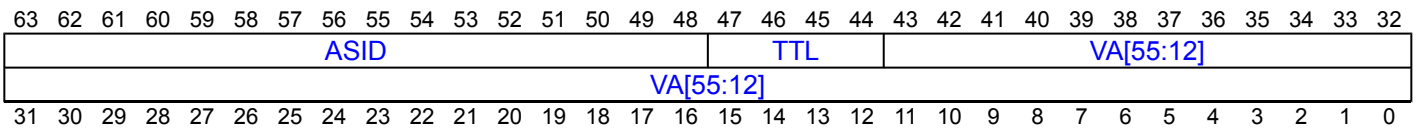
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VALE2OS, TLBI VALE2OSNXS are UNDEFINED.

Attributes

TLBI VALE2OS, TLBI VALE2OSNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.

- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VALE2OS, TLBI VALE2OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VALE2OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VALE2OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE3, TLBI VALE3NXS, TLB Invalidate by VA, Last level, EL3

The TLBI VALE3, TLBI VALE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

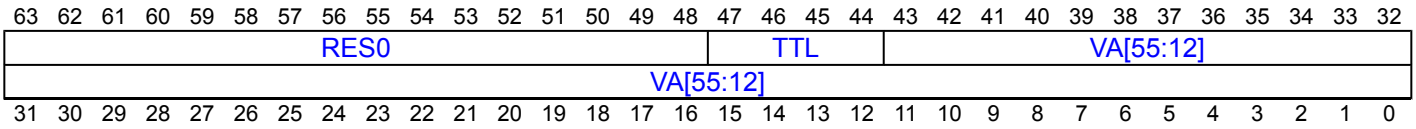
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VALE3, TLBI VALE3NXS are UNDEFINED.

Attributes

TLBI VALE3, TLBI VALE3NXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VALE3, TLBI VALE3NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VALE3{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0111	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VALE3NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0111	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE3IS, TLBI VALE3ISNXS, TLB Invalidate by VA, Last level, EL3, Inner Shareable

The TLBI VALE3IS, TLBI VALE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

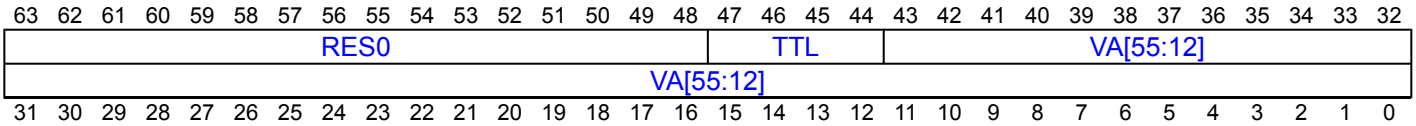
Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VALE3IS, TLBI VALE3ISNXS are UNDEFINED.

Attributes

TLBI VALE3IS, TLBI VALE3ISNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VALE3IS, TLBI VALE3ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VALE3IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0011	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VALE3ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0011	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE3OS, TLBI VALE3OSNXS, TLB Invalidate by VA, Last level, EL3, Outer Shareable

The TLBI VALE3OS, TLBI VALE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

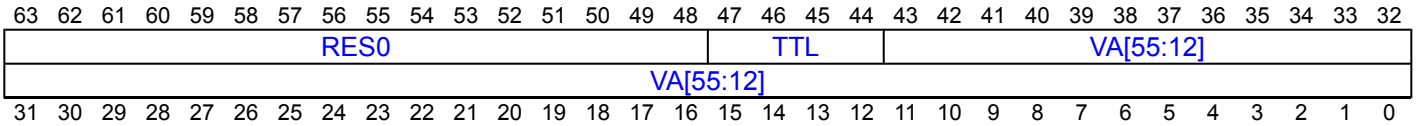
Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VALE3OS, TLBI VALE3OSNXS are UNDEFINED.

Attributes

TLBI VALE3OS, TLBI VALE3OSNXS is a 64-bit System instruction.

Field descriptions



Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing TLBI VALE3OS, TLBI VALE3OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VALE3OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VALE3OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLE1, TLBI VMALLE1NXS, TLB Invalidate by VMID, All at stage 1, EL1

The TLBI VMALLE1, TLBI VMALLE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VMALLE1, TLBI VMALLE1NXS are UNDEFINED.

Attributes

TLBI VMALLE1, TLBI VMALLE1NXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI VMALLE1, TLBI VMALLE1NXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VMALLE1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVMALLE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VMALLE1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVMALLE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLE1IS, TLBI VMALLE1ISNXS, TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable

The TLBI VMALLE1IS, TLBI VMALLE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VMALLE1IS, TLBI VMALLE1ISNXS are UNDEFINED.

Attributes

TLBI VMALLE1IS, TLBI VMALLE1ISNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI VMALLE1IS, TLBI VMALLE1ISNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VMALLE1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVMALLE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VMALLE1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVMALLE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLE1OS, TLBI VMALLE1OSNXS, TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable

The TLBI VMALLE1OS, TLBI VMALLE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VMALLE1OS, TLBI VMALLE1OSNXS are UNDEFINED.

Attributes

TLBI VMALLE1OS, TLBI VMALLE1OSNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI VMALLE1OS, TLBI VMALLE1OSNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VMALLE1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVMALLE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VMALLE1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVMALLE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLS12E1, TLBI VMALLS12E1NXS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1

The TLBI VMALLS12E1, TLBI VMALLS12E1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
  - If SCR_EL3.{NSE, NS} is {0, 0}, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {0, 1}, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {1, 1}, then:
    - The entry would be required to translate an address using the Realm EL1&0 translation regime.
    - The entry would be used with the current VMID.
- If FEAT_RME is not implemented, one of the following applies:
  - If SCR_EL3.NS is 0, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.NS is 1, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VMALLS12E1, TLBI VMALLS12E1NXS are UNDEFINED.

Attributes

TLBI VMALLS12E1, TLBI VMALLS12E1NXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI VMALLS12E1, TLBI VMALLS12E1NXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VMALLS12E1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0111	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID_NONE, Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VMALLS12E1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0111	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID_NONE, Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable

The TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
  - If SCR_EL3.{NSE, NS} is {0, 0}, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {0, 1}, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {1, 1}, then:
    - The entry would be required to translate an address using the Realm EL1&0 translation regime.
    - The entry would be used with the current VMID.
- If FEAT_RME is not implemented, one of the following applies:
  - If SCR_EL3.NS is 0, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.NS is 1, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS are UNDEFINED.

Attributes

TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VMALLS12E1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0011	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID_NONE, Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VMALLS12E1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0011	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID_NONE, Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable

The TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
  - If SCR_EL3.{NSE, NS} is {0, 0}, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {0, 1}, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {1, 1}, then:
    - The entry would be required to translate an address using the Realm EL1&0 translation regime.
    - The entry would be used with the current VMID.
- If FEAT_RME is not implemented, one of the following applies:
  - If SCR_EL3.NS is 0, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.NS is 1, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS are UNDEFINED.

Attributes

TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VMALLS12E1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0001	0b110

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID_NONE, Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VMALLS12E1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0001	0b110

```
if !(IsFeatureImplemented(FEAT_TLBIOS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID_NONE, Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLWS2E1, TLBI VMALLWS2E1NXS, TLB Invalidate stage 2 dirty state by VMID, EL1&0

The TLBI VMALLWS2E1, TLBI VMALLWS2E1NXS characteristics are:

Purpose

Invalidates stage 2 write permissions from cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry would be used for stage 2 translation. This applies if the TLB entry holds information from stage 2 translation only, or combined information from stage 1 and stage 2 translation.
- If FEAT_RME is implemented, one of the following applies:
  - If SCR_EL3.{NSE, NS} is {0, 0}, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {0, 1}, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {1, 1}, then:
    - The entry would be required to translate an address using the Realm EL1&0 translation regime.
    - The entry would be used with the current VMID.
- If FEAT_RME is not implemented, one of the following applies:
  - If SCR_EL3.NS is 0, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.NS is 1, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIW is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VMALLWS2E1, TLBI VMALLWS2E1NXS are UNDEFINED.

Attributes

TLBI VMALLWS2E1, TLBI VMALLWS2E1NXS is a 64-bit System instruction.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
RES0																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:0]

Reserved, RES0.

Executing TLBI VMALLWS2E1, TLBI VMALLWS2E1NXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VMALLWS2E1{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0110	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIW) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VMALLWS2E1NXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0110	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIW) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLWS2E1IS, TLBI VMALLWS2E1ISNXS, TLB Invalidate stage 2 dirty state by VMID, EL1&0, Inner Shareable

The TLBI VMALLWS2E1IS, TLBI VMALLWS2E1ISNXS characteristics are:

Purpose

Invalidates stage 2 write permissions from cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry would be used for stage 2 translation. This applies if the TLB entry holds information from stage 2 translation only, or combined information from stage 1 and stage 2 translation.
- If FEAT_RME is implemented, one of the following applies:
  - If SCR_EL3.{NSE, NS} is {0, 0}, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {0, 1}, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {1, 1}, then:
    - The entry would be required to translate an address using the Realm EL1&0 translation regime.
    - The entry would be used with the current VMID.
- If FEAT_RME is not implemented, one of the following applies:
  - If SCR_EL3.NS is 0, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.NS is 1, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

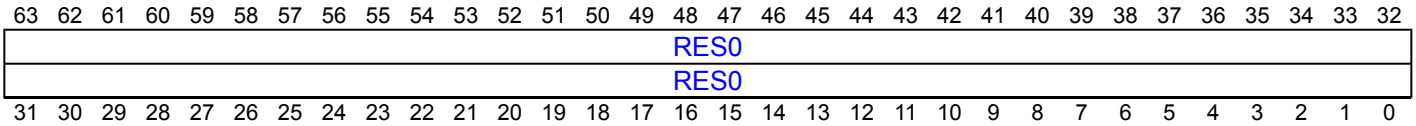
Configuration

This instruction is present only when FEAT_TLBIW is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VMALLWS2E1IS, TLBI VMALLWS2E1ISNXS are UNDEFINED.

Attributes

TLBI VMALLWS2E1IS, TLBI VMALLWS2E1ISNXS is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Reserved, RES0.

Executing TLBI VMALLWS2E1IS, TLBI VMALLWS2E1ISNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VMALLWS2E1IS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIW) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VMALLWS2E1ISNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIW) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLWS2E1OS, TLBI VMALLWS2E1OSNXS, TLB Invalidate stage 2 write permission by VMID, EL1&0, Outer Shareable

The TLBI VMALLWS2E1OS, TLBI VMALLWS2E1OSNXS characteristics are:

Purpose

Invalidates stage 2 write permissions from cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry would be used for stage 2 translation. This applies if the TLB entry holds information from stage 2 translation only, or combined information from stage 1 and stage 2 translation.
- If FEAT_RME is implemented, one of the following applies:
  - If SCR_EL3.{NSE, NS} is {0, 0}, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {0, 1}, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.
  - If SCR_EL3.{NSE, NS} is {1, 1}, then:
    - The entry would be required to translate an address using the Realm EL1&0 translation regime.
    - The entry would be used with the current VMID.
- If FEAT_RME is not implemented, one of the following applies:
  - If SCR_EL3.NS is 0, then:
    - The entry would be required to translate an address using the Secure EL1&0 translation regime.
    - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
  - If SCR_EL3.NS is 1, then:
    - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
    - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

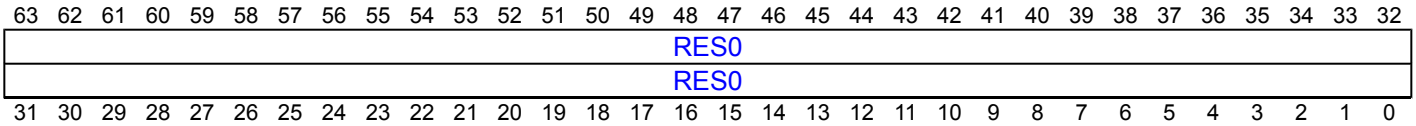
Configuration

This instruction is present only when FEAT_TLBIW is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBI VMALLWS2E1OS, TLBI VMALLWS2E1OSNXS are UNDEFINED.

Attributes

TLBI VMALLWS2E1OS, TLBI VMALLWS2E1OSNXS is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Reserved, RES0.

Executing TLBI VMALLWS2E1OS, TLBI VMALLWS2E1OSNXS

The Rt field should be set to `0b11111`. If the Rt field is not set to `0b11111`, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to `0b11111`.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI VMALLWS2E1OS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0101	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIW) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_AllAttr, X[t, 64]);
```

When FEAT_XS is implemented

```
TLBI VMALLWS2E1OSNXS{, <Xt>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0101	0b010

```
if !(IsFeatureImplemented(FEAT_TLBIW) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBI_VMALLWS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBI_ExcludeXS, X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP IPAS2E1, TLBIP IPAS2E1NXS, TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1

The TLBIP IPAS2E1, TLBIP IPAS2E1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from any level of the translation table walk.
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

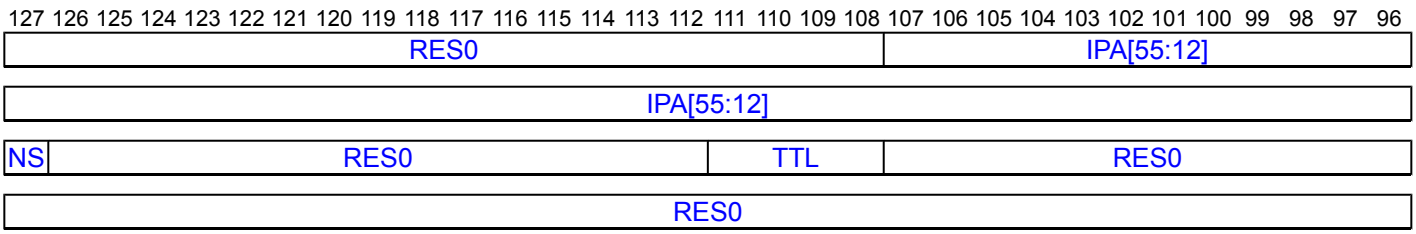
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP IPAS2E1, TLBIP IPAS2E1NXS are UNDEFINED.

Attributes

TLBIP IPAS2E1, TLBIP IPAS2E1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

IPA[55:12], bits [107:64]

Bits[55:12] of the intermediate physical address to match.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP IPAS2E1, TLBIP IPAS2E1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP IPAS2E1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP IPAS2E1NXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP IPAS2E1IS, TLBIP IPAS2E1ISNXS, TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

The TLBIP IPAS2E1IS, TLBIP IPAS2E1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from any level of the translation table walk.
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

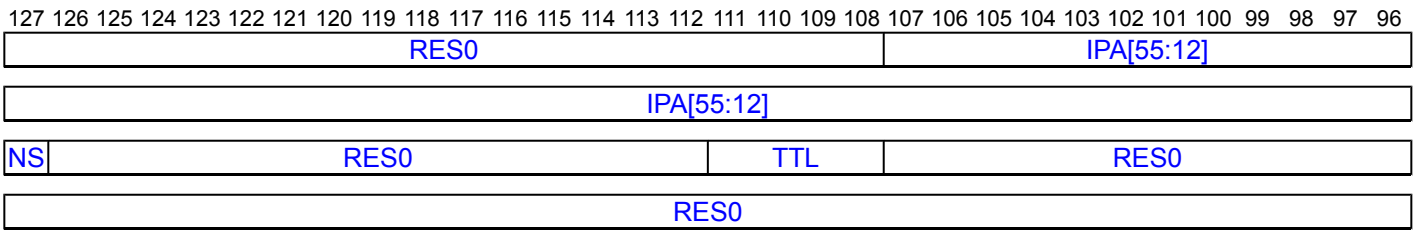
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP IPAS2E1IS, TLBIP IPAS2E1ISNXS are UNDEFINED.

Attributes

TLBIP IPAS2E1IS, TLBIP IPAS2E1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

IPA[55:12], bits [107:64]

Bits[55:12] of the intermediate physical address to match.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP IPAS2E1IS, TLBIP IPAS2E1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP IPAS2E1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP IPAS2E1ISNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP IPAS2E1OS, TLBIP IPAS2E1OSNXS, TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

The TLBIP IPAS2E1OS, TLBIP IPAS2E1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from any level of the translation table walk.
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

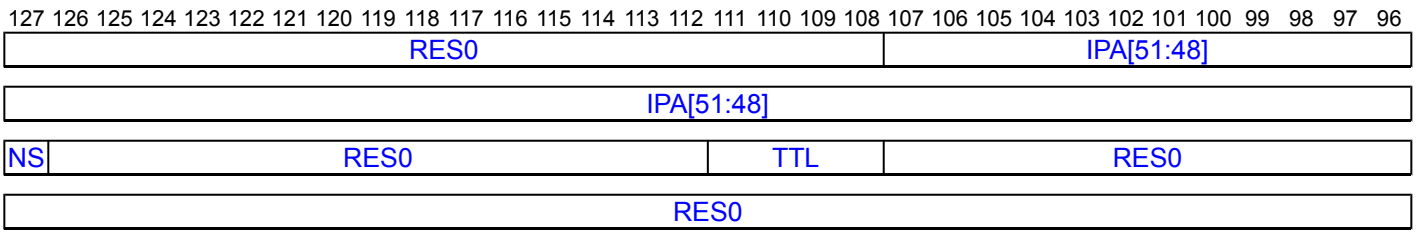
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP IPAS2E1OS, TLBIP IPAS2E1OSNXS are UNDEFINED.

Attributes

TLBIP IPAS2E1OS, TLBIP IPAS2E1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

IPA[51:48], bits [107:64]

Bits[55:12] of the intermediate physical address to match.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP IPAS2E1OS, TLBIP IPAS2E1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP IPAS2E1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Any,
TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP IPAS2E1OSNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH, TLBILevel_Any,
TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP IPAS2LE1, TLBIP IPAS2LE1NXS, TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1

The TLBIP IPAS2LE1, TLBIP IPAS2LE1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from the final level of the translation table walk.
  - A 64-bit stage 2 only translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

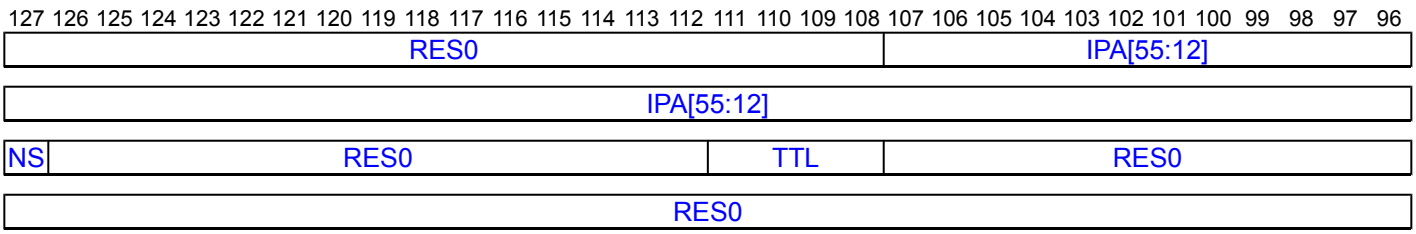
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP IPAS2LE1, TLBIP IPAS2LE1NXS are UNDEFINED.

Attributes

TLBIP IPAS2LE1, TLBIP IPAS2LE1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

IPA[55:12], bits [107:64]

Bits[55:12] of the intermediate physical address to match.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP IPAS2LE1, TLBIP IPAS2LE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP IPAS2LE1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP IPAS2LE1NXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP IPAS2LE1IS, TLBIP IPAS2LE1ISNXS, TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

The TLBIP IPAS2LE1IS, TLBIP IPAS2LE1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from the final level of the translation table walk.
  - A 64-bit stage 2 only translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

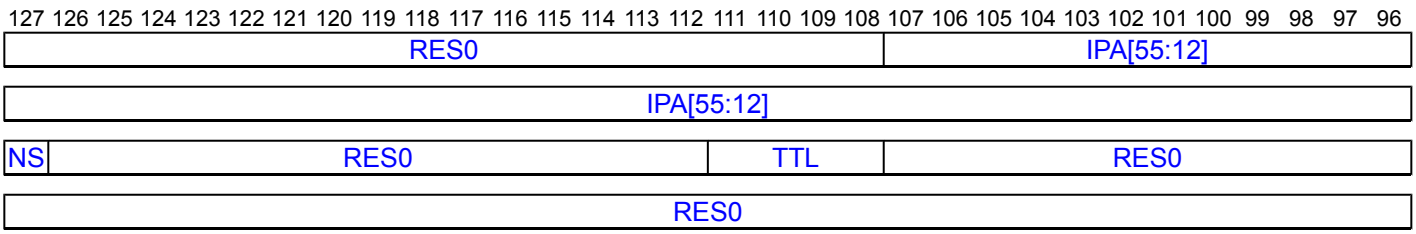
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP IPAS2LE1IS, TLBIP IPAS2LE1ISNXS are UNDEFINED.

Attributes

TLBIP IPAS2LE1IS, TLBIP IPAS2LE1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

IPA[55:12], bits [107:64]

Bits[55:12] of the intermediate physical address to match.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP IPAS2LE1IS, TLBIP IPAS2LE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP IPAS2LE1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP IPAS2LE1ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP IPAS2LE1OS, TLBIP IPAS2LE1OSNXS, TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

The TLBIP IPAS2LE1OS, TLBIP IPAS2LE1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from the final level of the translation table walk.
  - A 64-bit stage 2 only translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

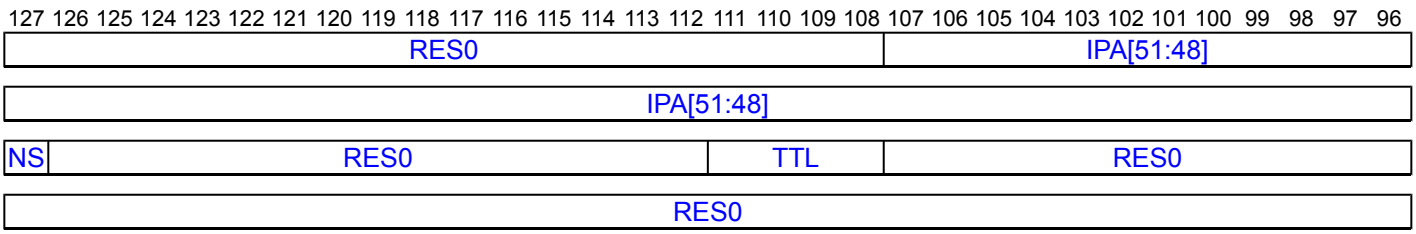
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP IPAS2LE1OS, TLBIP IPAS2LE1OSNXS are UNDEFINED.

Attributes

TLBIP IPAS2LE1OS, TLBIP IPAS2LE1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

IPA[51:48], bits [107:64]

Bits[55:12] of the intermediate physical address to match.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP IPAS2LE1OS, TLBIP IPAS2LE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP IPAS2LE1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b100

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP IPAS2LE1OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b100

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RIPAS2E1, TLBIP RIPAS2E1NXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1

The TLBIP RIPAS2E1, TLBIP RIPAS2E1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

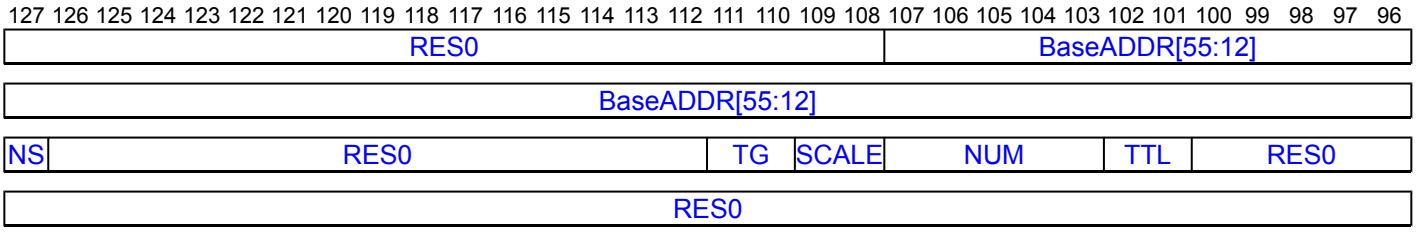
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RIPAS2E1, TLBIP RIPAS2E1NXS are UNDEFINED.

Attributes

TLBIP RIPAS2E1, TLBIP RIPAS2E1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RIPAS2E1, TLBIP RIPAS2E1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RIPAS2E1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP RIPAS2E1NXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RIPAS2E1IS, TLBIP RIPAS2E1ISNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

The TLBIP RIPAS2E1IS, TLBIP RIPAS2E1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

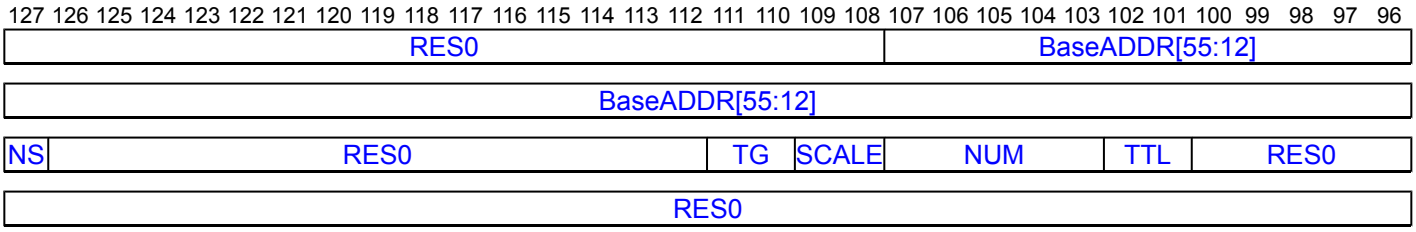
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RIPAS2E1IS, TLBIP RIPAS2E1ISNXS are UNDEFINED.

Attributes

TLBIP RIPAS2E1IS, TLBIP RIPAS2E1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RIPAS2E1IS, TLBIP RIPAS2E1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RIPAS2E1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP RIPAS2E1ISNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RIPAS2E1OS, TLBIP RIPAS2E1OSNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

The TLBIP RIPAS2E1OS, TLBIP RIPAS2E1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 2 only translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

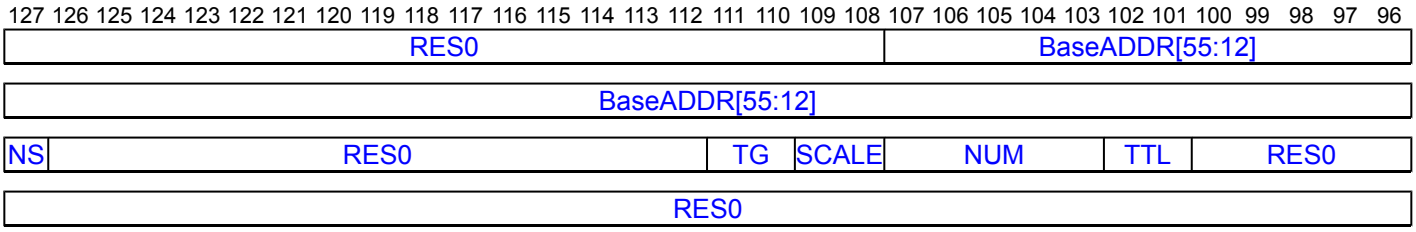
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RIPAS2E1OS, TLBIP RIPAS2E1OSNXS are UNDEFINED.

Attributes

TLBIP RIPAS2E1OS, TLBIP RIPAS2E1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RIPAS2E1OS, TLBIP RIPAS2E1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RIPAS2E1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RIPAS2E1OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RIPAS2LE1, TLBIP RIPAS2LE1NXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

The TLBIP RIPAS2LE1, TLBIP RIPAS2LE1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - If FEAT_D128 is implemented, a 64-bit stage 2 only translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE+1)}$ * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

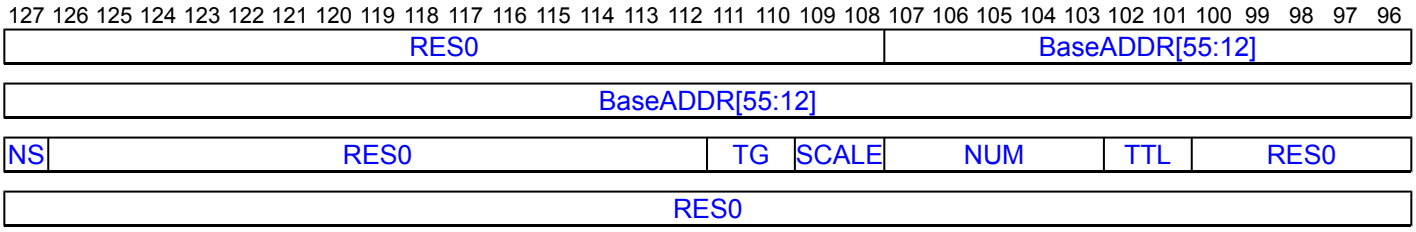
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RIPAS2LE1, TLBIP RIPAS2LE1NXS are UNDEFINED.

Attributes

TLBIP RIPAS2LE1, TLBIP RIPAS2LE1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RIPAS2LE1, TLBIP RIPAS2LE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RIPAS2LE1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b110

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP RIPAS2LE1NXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b110

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RIPAS2LE1IS, TLBIP RIPAS2LE1ISNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

The TLBIP RIPAS2LE1IS, TLBIP RIPAS2LE1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - If FEAT_D128 is implemented, a 64-bit stage 2 only translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

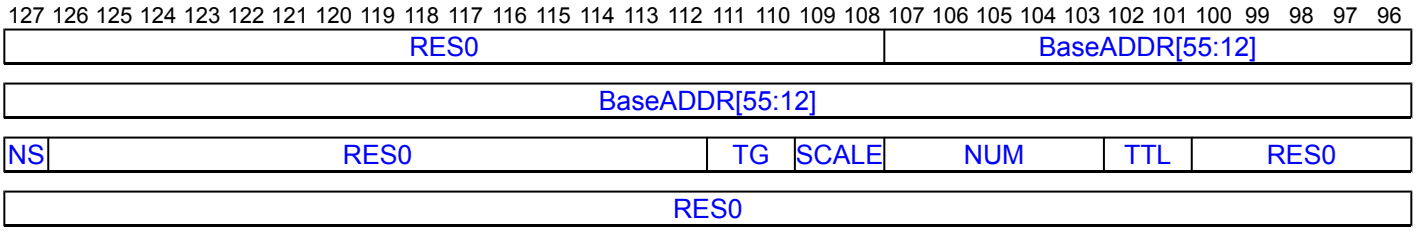
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RIPAS2LE1IS, TLBIP RIPAS2LE1ISNXS are UNDEFINED.

Attributes

TLBIP RIPAS2LE1IS, TLBIP RIPAS2LE1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RIPAS2LE1IS, TLBIP RIPAS2LE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RIPAS2LE1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP RIPAS2LE1ISNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RIPAS2LE1OS, TLBIP RIPAS2LE1OSNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

The TLBIP RIPAS2LE1OS, TLBIP RIPAS2LE1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 2 only translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - If FEAT_D128 is implemented, a 64-bit stage 2 only translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- If FEAT_RME is implemented, one of the following applies:
  - SCR_EL3.{NSE, NS} is {0, 0} and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {0, 1} and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
  - SCR_EL3.{NSE, NS} is {1, 1} and the entry would be required to translate any IPA in the specified address range using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
  - SCR_EL3.NS is 0 and the entry would be required to translate any IPA in the specified address range using the Secure EL1&0 translation regime.
  - SCR_EL3.NS is 1 and the entry would be required to translate any IPA in the specified address range using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

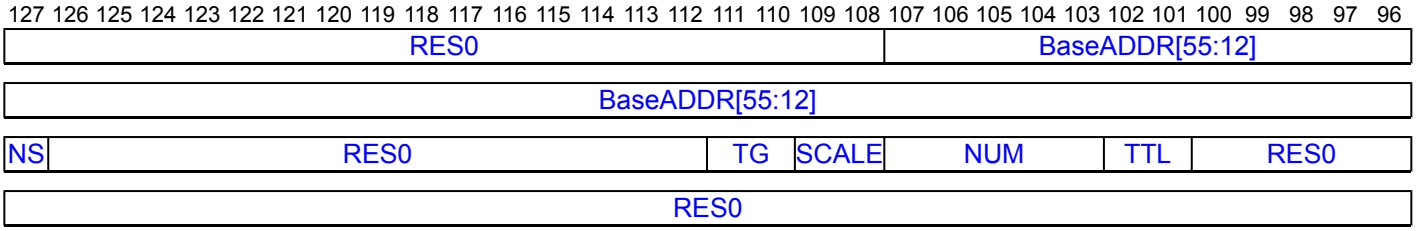
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RIPAS2LE1OS, TLBIP RIPAS2LE1OSNXS are UNDEFINED.

Attributes

TLBIP RIPAS2LE1OS, TLBIP RIPAS2LE1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

NS	Meaning
0b0	IPA is in the Secure IPA space.
0b1	IPA is in the Non-secure IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RIPAS2LE1OS, TLBIP RIPAS2LE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RIPAS2LE1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RIPAS2LE1OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        return;
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAAE1, TLBIP RVAAE1NXS, TLB Range Invalidate by VA, All ASID, EL1

The TLBIP RVAAE1, TLBIP RVAAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

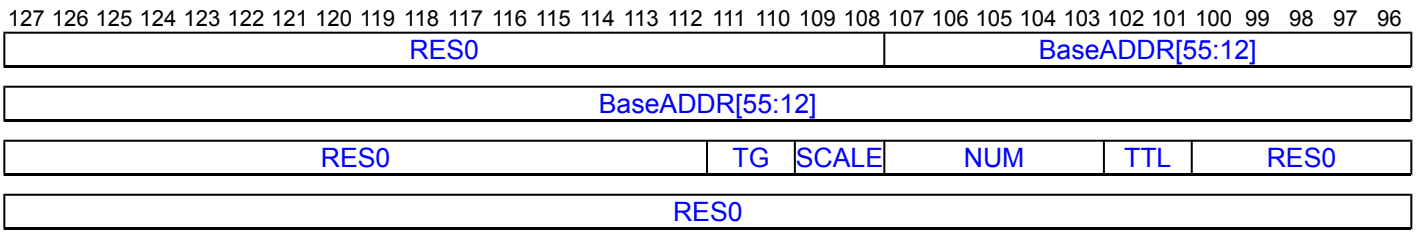
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAAE1, TLBIP RVAAE1NXS are UNDEFINED.

Attributes

TLBIP RVAAE1, TLBIP RVAAE1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAAE1, TLBIP RVAAE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAAE1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0110	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAAE1NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2

0b01	0b000	0b1001	0b0110	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAAE1IS, TLBIP RVAAE1ISNXS, TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable

The TLBIP RVAAE1IS, TLBIP RVAAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

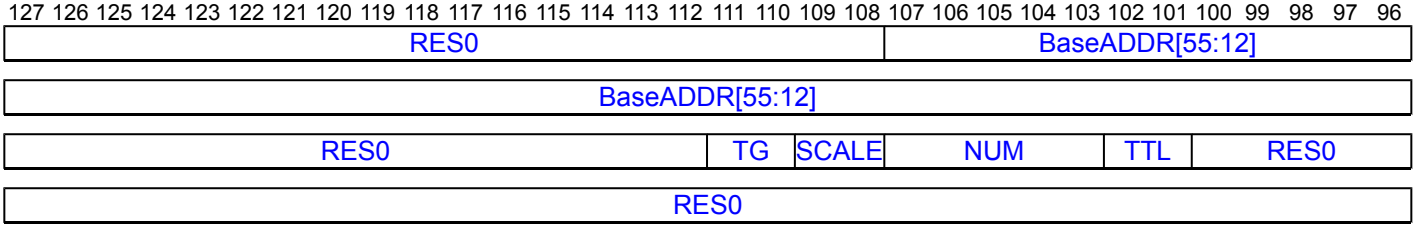
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAAE1IS, TLBIP RVAAE1ISNXS are UNDEFINED.

Attributes

TLBIP RVAAE1IS, TLBIP RVAAE1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.

- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAAE1IS, TLBIP RVAAE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAAE1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAAE1ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAAE1OS, TLBIP RVAAE1OSNXS, TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

The TLBIP RVAAE1OS, TLBIP RVAAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

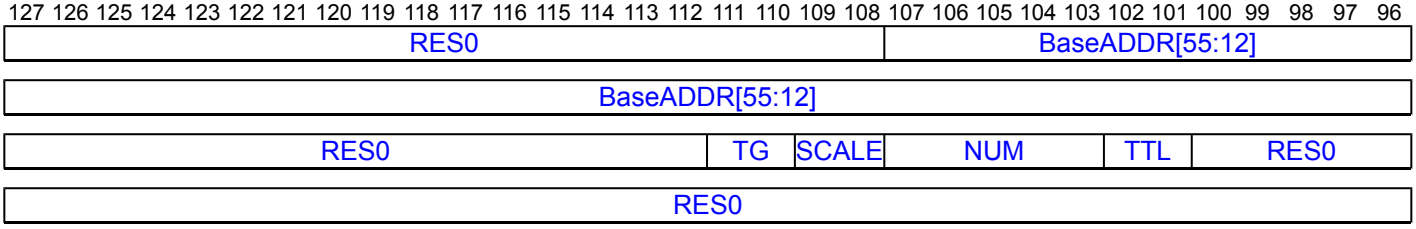
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAAE1OS, TLBIP RVAAE1OSNXS are UNDEFINED.

Attributes

TLBIP RVAAE1OS, TLBIP RVAAE1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.

- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAAE1OS, TLBIP RVAAE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAAE1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0101	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAAE1OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0101	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAALE1, TLBIP RVAALE1NXS, TLB Range Invalidate by VA, All ASID, Last level, EL1

The TLBIP RVAALE1, TLBIP RVAALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - A 64-bit stage 1 translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

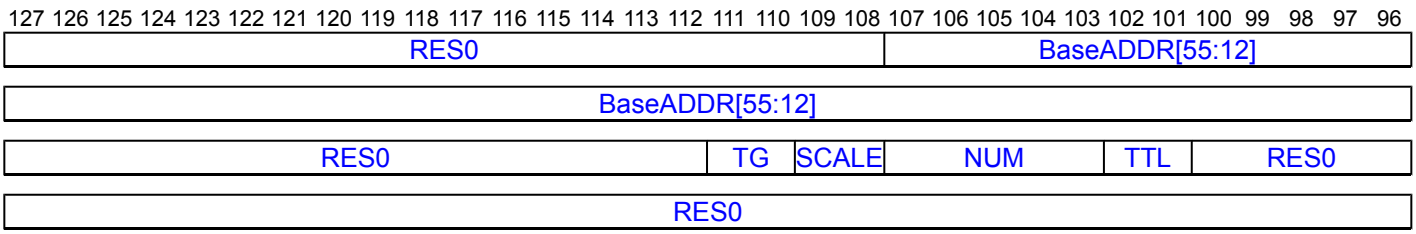
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAALE1, TLBIP RVAALE1NXS are UNDEFINED.

Attributes

TLBIP RVAALE1, TLBIP RVAALE1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAALE1, TLBIP RVAALE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAALE1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0110	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAALE1NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2

0b01	0b000	0b1001	0b0110	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAALE1IS, TLBIP RVAALE1ISNXS, TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

The TLBIP RVAALE1IS, TLBIP RVAALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - A 64-bit stage 1 translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE+1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

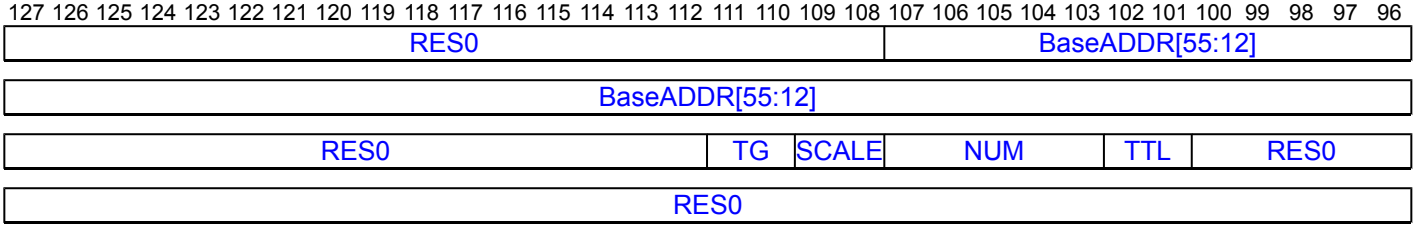
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAALE1IS, TLBIP RVAALE1ISNXS are UNDEFINED.

Attributes

TLBIP RVAALE1IS, TLBIP RVAALE1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.

- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAALE1IS, TLBIP RVAALE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAALE1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0010	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAALE1ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0010	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAALE1OS, TLBIP RVAALE1OSNXS, TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

The TLBIP RVAALE1OS, TLBIP RVAALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from the leaf level of the translation table walk, indicated by the TTL hint.
  - A 64-bit stage 1 translation table entry, from the leaf level of the translation table walk, if TTL is `0b00`.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE+1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

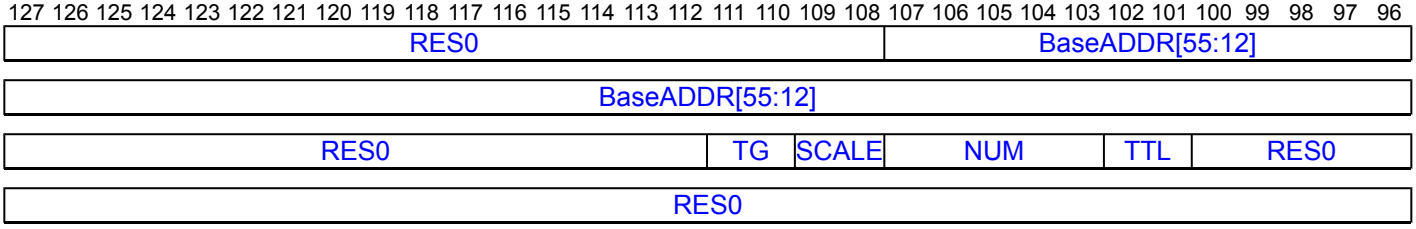
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAALE1OS, TLBIP RVAALE1OSNXS are UNDEFINED.

Attributes

TLBIP RVAALE1OS, TLBIP RVAALE1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.

- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAALE1OS, TLBIP RVAALE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAALE1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0101	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAALE1OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0101	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAE1, TLBIP RVAE1NXS, TLB Range Invalidate by VA, EL1

The TLBIP RVAE1, TLBIP RVAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

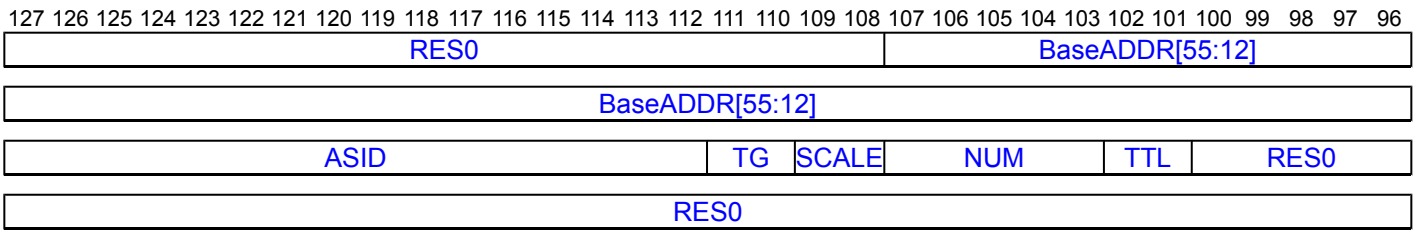
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAE1, TLBIP RVAE1NXS are UNDEFINED.

Attributes

TLBIP RVAE1, TLBIP RVAE1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAE1, TLBIP RVAE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAE1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAE1NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAE1IS, TLBIP RVAE1ISNXS, TLB Range Invalidate by VA, EL1, Inner Shareable

The TLBIP RVAE1IS, TLBIP RVAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

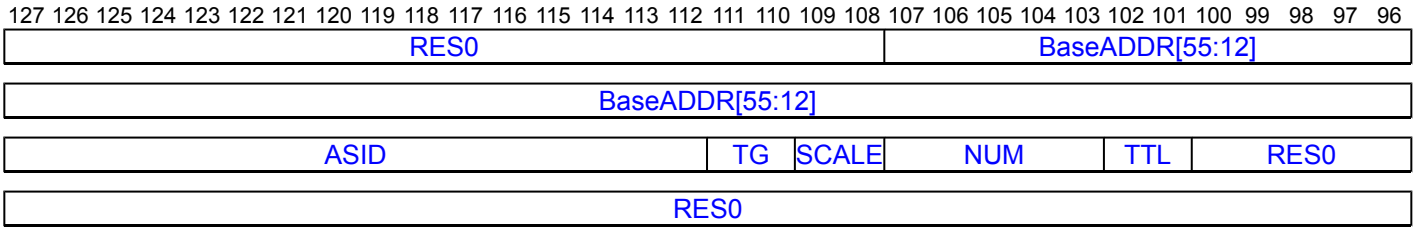
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAE1IS, TLBIP RVAE1ISNXS are UNDEFINED.

Attributes

TLBIP RVAE1IS, TLBIP RVAE1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAE1IS, TLBIP RVAE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAE1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAE1ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2

0b01	0b000	0b1001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAE1OS, TLBIP RVAE1OSNXS, TLB Range Invalidate by VA, EL1, Outer Shareable

The TLBIP RVAE1OS, TLBIP RVAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

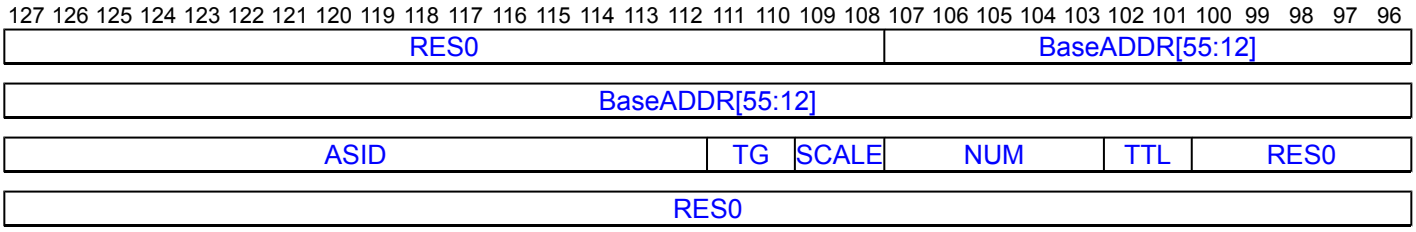
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAE1OS, TLBIP RVAE1OSNXS are UNDEFINED.

Attributes

TLBIP RVAE1OS, TLBIP RVAE1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAE1OS, TLBIP RVAE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAE1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAE1OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2

0b01	0b000	0b1001	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAE2, TLBIP RVAE2NXS, TLB Range Invalidate by VA, EL2

The TLBIP RVAE2, TLBIP RVAE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

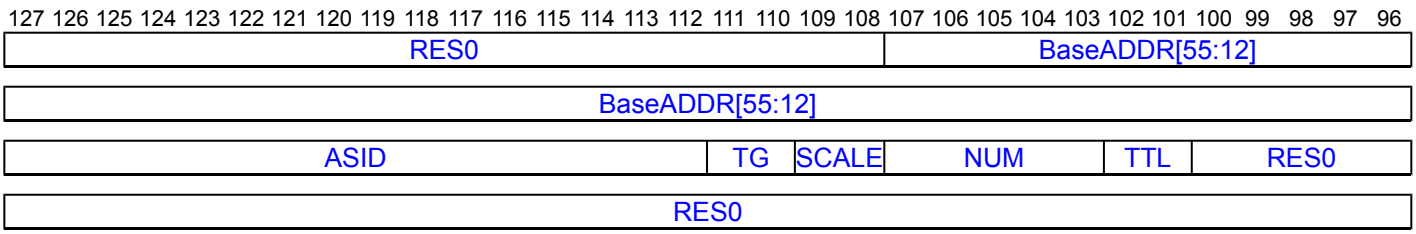
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAE2, TLBIP RVAE2NXS are UNDEFINED.

Attributes

TLBIP RVAE2, TLBIP RVAE2NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.

- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAE2, TLBIP RVAE2NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAE2{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAE2NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAE2IS, TLBIP RVAE2ISNXS, TLB Range Invalidate by VA, EL2, Inner Shareable

The TLBIP RVAE2IS, TLBIP RVAE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

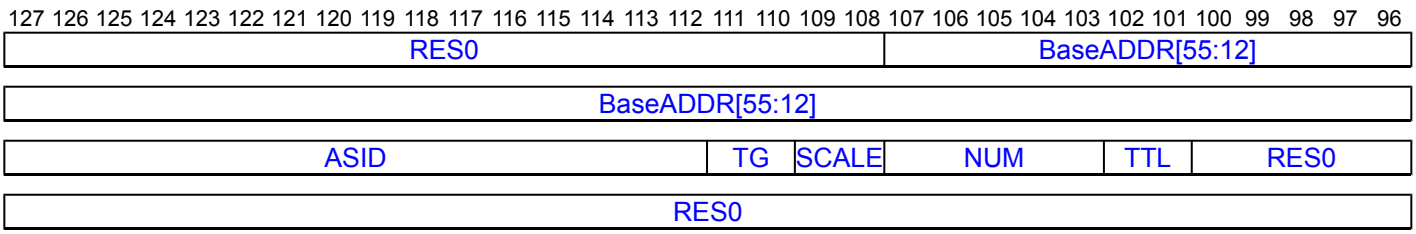
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAE2IS, TLBIP RVAE2ISNXS are UNDEFINED.

Attributes

TLBIP RVAE2IS, TLBIP RVAE2ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.

- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAE2IS, TLBIP RVAE2ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAE2IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAE2ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAE2OS, TLBIP RVAE2OSNXS, TLB Range Invalidate by VA, EL2, Outer Shareable

The TLBIP RVAE2OS, TLBIP RVAE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

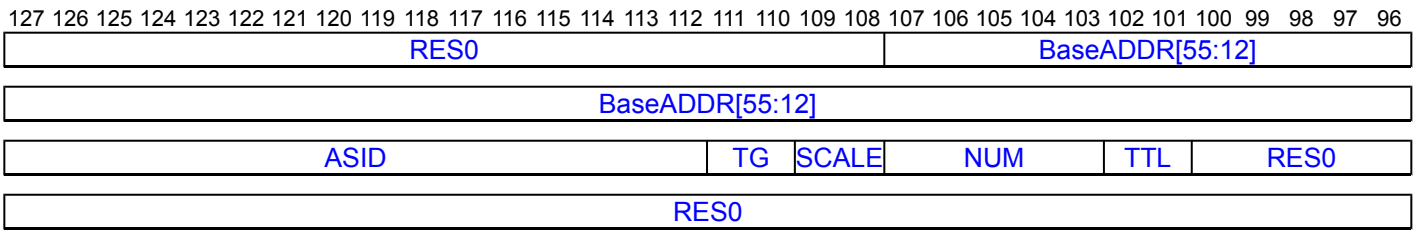
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAE2OS, TLBIP RVAE2OSNXS are UNDEFINED.

Attributes

TLBIP RVAE2OS, TLBIP RVAE2OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.

- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAE2OS, TLBIP RVAE2OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAE2OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAE2OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAE3, TLBIP RVAE3NXS, TLB Range Invalidate by VA, EL3

The TLBIP RVAE3, TLBIP RVAE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

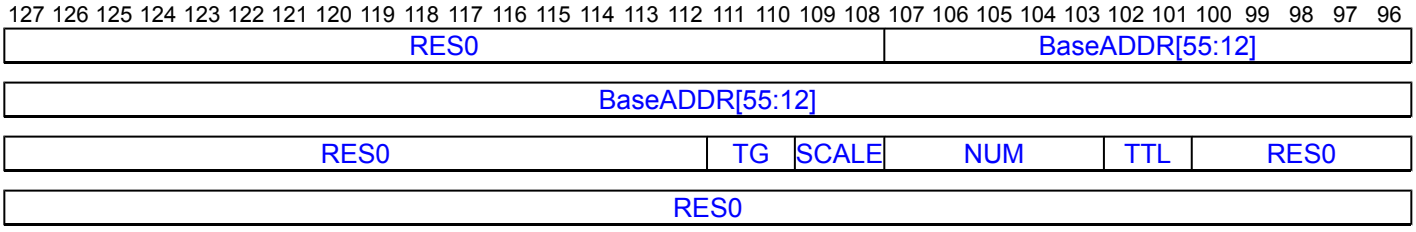
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAE3, TLBIP RVAE3NXS are UNDEFINED.

Attributes

TLBIP RVAE3, TLBIP RVAE3NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAE3, TLBIP RVAE3NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAE3{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP RVAE3NXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0110	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAE3IS, TLBIP RVAE3ISNXS, TLB Range Invalidate by VA, EL3, Inner Shareable

The TLBIP RVAE3IS, TLBIP RVAE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

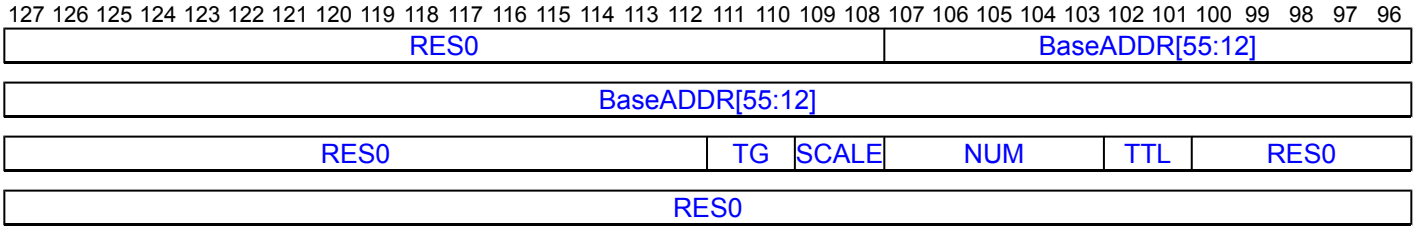
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAE3IS, TLBIP RVAE3ISNXS are UNDEFINED.

Attributes

TLBIP RVAE3IS, TLBIP RVAE3ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAE3IS, TLBIP RVAE3ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAE3IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP RVAE3ISNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVAE3OS, TLBIP RVAE3OSNXS, TLB Range Invalidate by VA, EL3, Outer Shareable

The TLBIP RVAE3OS, TLBIP RVAE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

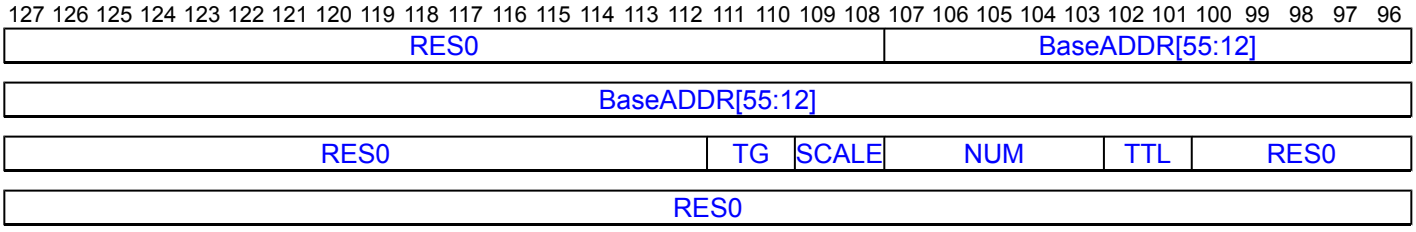
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVAE3OS, TLBIP RVAE3OSNXS are UNDEFINED.

Attributes

TLBIP RVAE3OS, TLBIP RVAE3OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVAE3OS, TLBIP RVAE3OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVAE3OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVAE3OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVALE1, TLBIP RVALE1NXS, TLB Range Invalidate by VA, Last level, EL1

The TLBIP RVALE1, TLBIP RVALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

For more information about the architectural requirements for this System instruction, see 'Invalidating TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

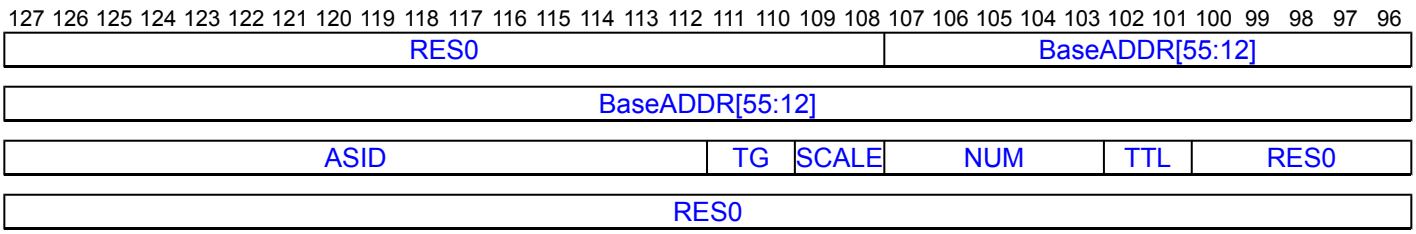
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVALE1, TLBIP RVALE1NXS are UNDEFINED.

Attributes

TLBIP RVALE1, TLBIP RVALE1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVALE1, TLBIP RVALE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVALE1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVALE1NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVALE1IS, TLBIP RVALE1ISNXS, TLB Range Invalidate by VA, Last level, EL1, Inner Shareable

The TLBIP RVALE1IS, TLBIP RVALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

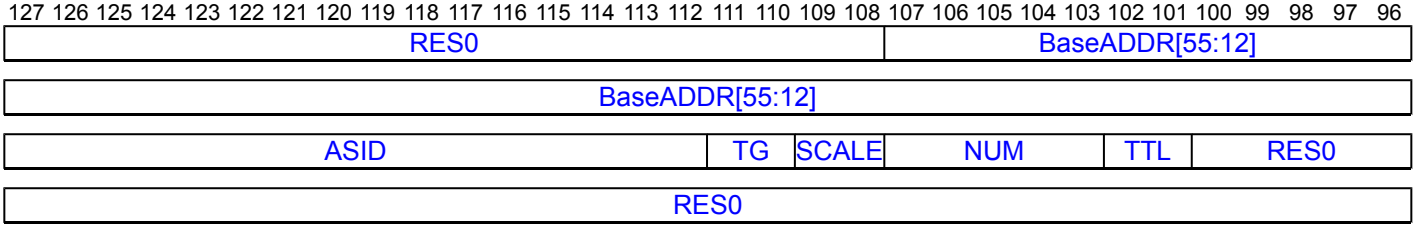
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVALE1IS, TLBIP RVALE1ISNXS are UNDEFINED.

Attributes

TLBIP RVALE1IS, TLBIP RVALE1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVALE1IS, TLBIP RVALE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVALE1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVALE1ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVALE1OS, TLBIP RVALE1OSNXS, TLB Range Invalidate by VA, Last level, EL1, Outer Shareable

The TLBIP RVALE1OS, TLBIP RVALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate any of the VAs in the specified address range using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate any of the VAs in the specified address range using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

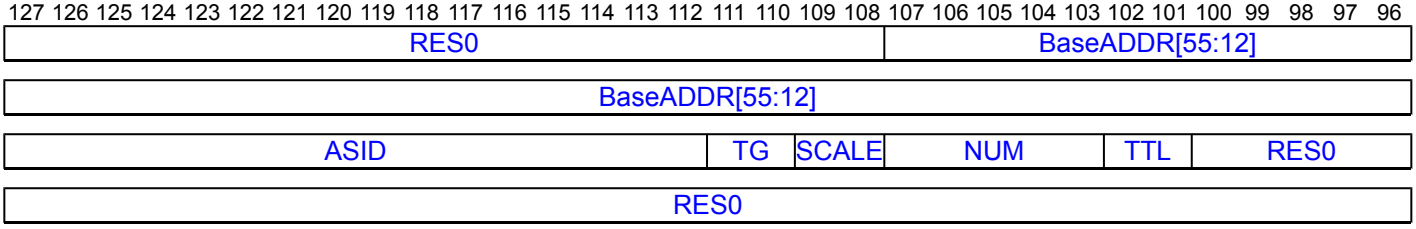
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVALE1OS, TLBIP RVALE1OSNXS are UNDEFINED.

Attributes

TLBIP RVALE1OS, TLBIP RVALE1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVALE1OS, TLBIP RVALE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVALE1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIRVALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP RVALE1OSNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIRVALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVALE2, TLBIP RVALE2NXS, TLB Range Invalidate by VA, Last level, EL2

The TLBIP RVALE2, TLBIP RVALE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

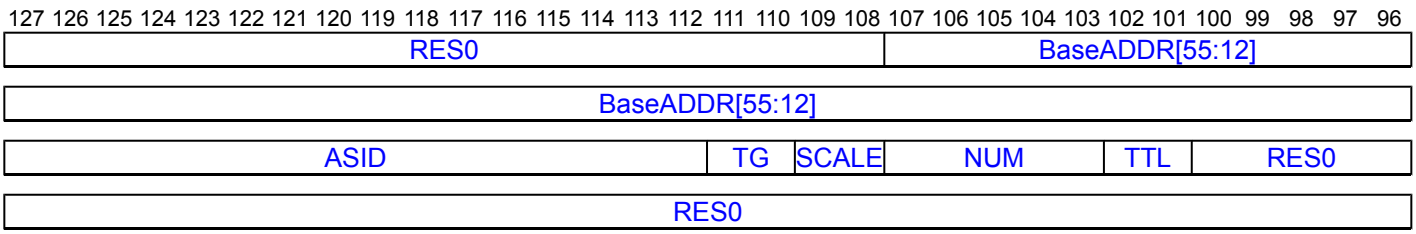
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVALE2, TLBIP RVALE2NXS are UNDEFINED.

Attributes

TLBIP RVALE2, TLBIP RVALE2NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.

- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVALE2, TLBIP RVALE2NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVALE2{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVALE2NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVALE2IS, TLBIP RVALE2ISNXS, TLB Range Invalidate by VA, Last level, EL2, Inner Shareable

The TLBIP RVALE2IS, TLBIP RVALE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

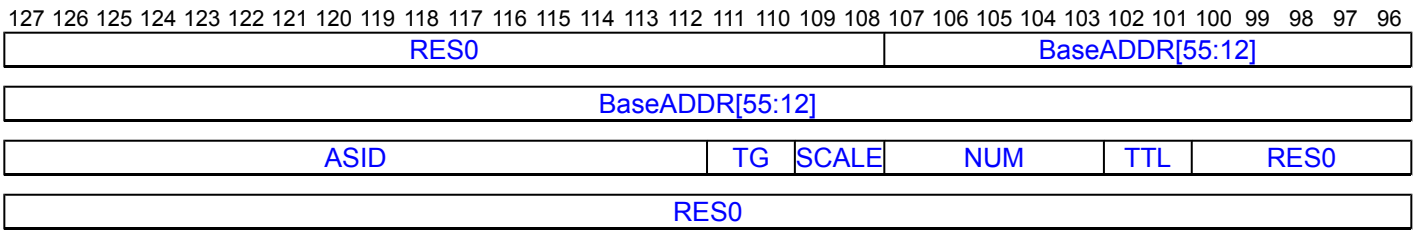
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVALE2IS, TLBIP RVALE2ISNXS are UNDEFINED.

Attributes

TLBIP RVALE2IS, TLBIP RVALE2ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.

- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVALE2IS, TLBIP RVALE2ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVALE2IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVALE2ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVALE2OS, TLBIP RVALE2OSNXS, TLB Range Invalidate by VA, Last level, EL2, Outer Shareable

The TLBIP RVALE2OS, TLBIP RVALE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, if TTL is `0b00`.
- The entry would be used to translate any VA in the range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)] using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

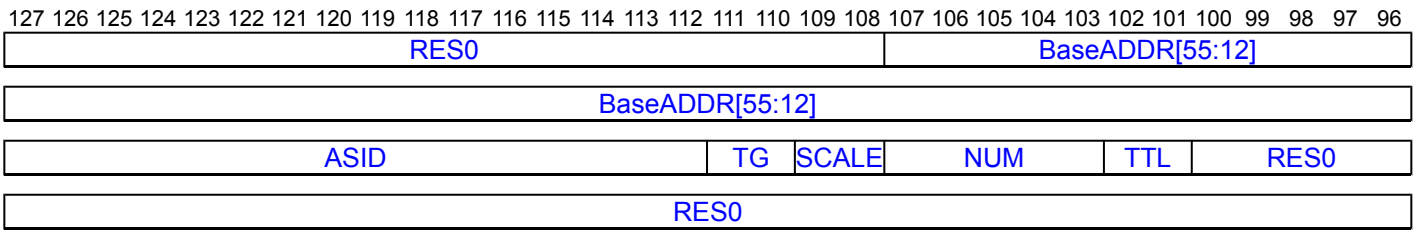
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVALE2OS, TLBIP RVALE2OSNXS are UNDEFINED.

Attributes

TLBIP RVALE2OS, TLBIP RVALE2OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.

- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVALE2OS, TLBIP RVALE2OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVALE2OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP RVALE2OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVALE3, TLBIP RVALE3NXS, TLB Range Invalidate by VA, Last level, EL3

The TLBIP RVALE3, TLBIP RVALE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from the final level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

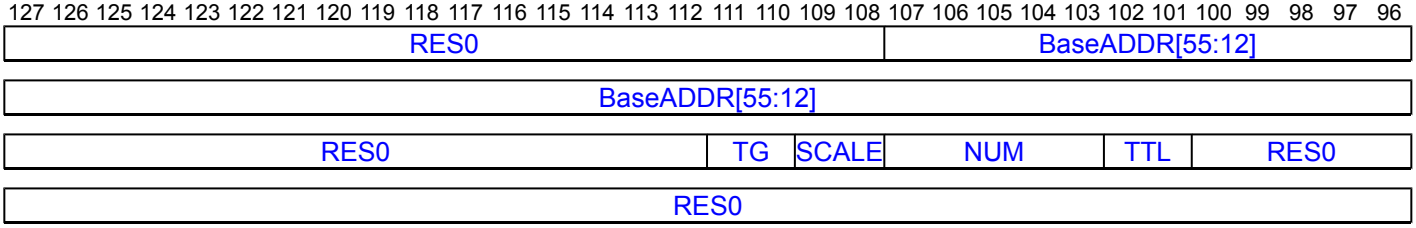
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVALE3, TLBIP RVALE3NXS are UNDEFINED.

Attributes

TLBIP RVALE3, TLBIP RVALE3NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVALE3, TLBIP RVALE3NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVALE3{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP RVALE3NXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0110	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVALE3IS, TLBIP RVALE3ISNXS, TLB Range Invalidate by VA, Last level, EL3, Inner Shareable

The TLBIP RVALE3IS, TLBIP RVALE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from the final level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

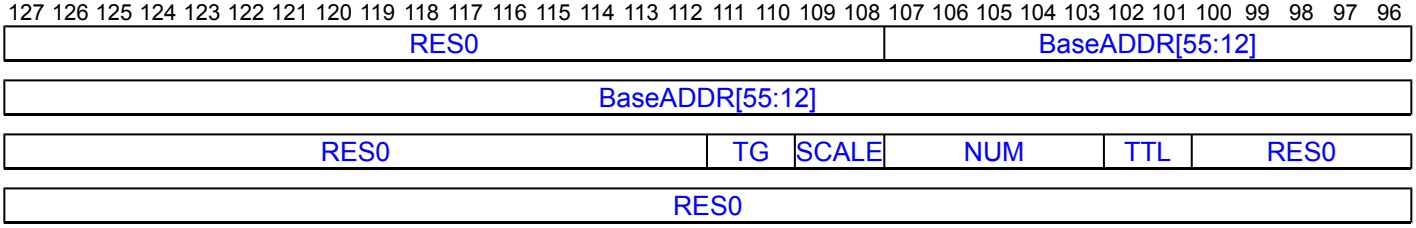
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVALE3IS, TLBIP RVALE3ISNXS are UNDEFINED.

Attributes

TLBIP RVALE3IS, TLBIP RVALE3ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVALE3IS, TLBIP RVALE3ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVALE3IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP RVALE3ISNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP RVALE3OS, TLBIP RVALE3OSNXS, TLB Range Invalidate by VA, Last level, EL3, Outer Shareable

The TLBIP RVALE3OS, TLBIP RVALE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from the final level of the translation table walk up to the level indicated in the TTL hint.
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL is `0b00`.
- The entry would be used to translate any of the VAs in the specified address range using the EL3 translation regime.
- The entry is within the address range determined by the formula [BaseADDR <= VA < BaseADDR + ((NUM +1)*$2^{(5*SCALE +1)}$ * Translation_Granule_Size)].

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For 128-bit translation table entry, the range of addresses invalidated is UNPREDICTABLE when Block or Page size corresponding to TTL and TG, for the translation system is not aligned.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

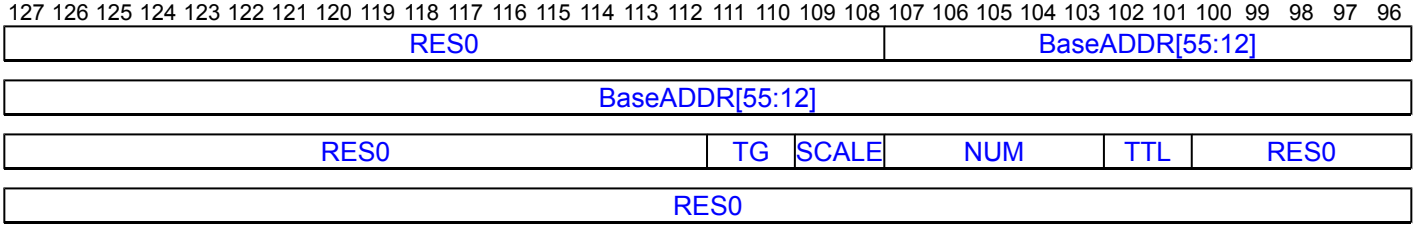
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP RVALE3OS, TLBIP RVALE3OSNXS are UNDEFINED.

Attributes

TLBIP RVALE3OS, TLBIP RVALE3OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

BaseADDR[55:12], bits [107:64]

The starting address for the range of the maintenance instructions. This field is BaseADDR[55:12] for all translation granules.

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

TG	Meaning
0b00	Reserved.
0b01	4K translation granule.
0b10	16K translation granule.
0b11	64K translation granule.

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate:

- Non-leaf-level entries in the range up to but not including the level described by the TTL hint.
- Leaf-level entries in the range that match the level described by the TTL hint.

TTL	Meaning
0b00	The entries in the range can be using any level for the translation table entries.
0b01	The TTL hint indicates level 1.
0b10	The TTL hint indicates level 2.
0b11	The TTL hint indicates level 3.

Bits [36:0]

Reserved, RES0.

Executing TLBIP RVALE3OS, TLBIP RVALE3OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP RVALE3OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP RVALE3OSNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAAE1, TLBIP VAAE1NXS, TLB Invalidate Pair by VA, All ASID, EL1

The TLBIP VAAE1, TLBIP VAAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

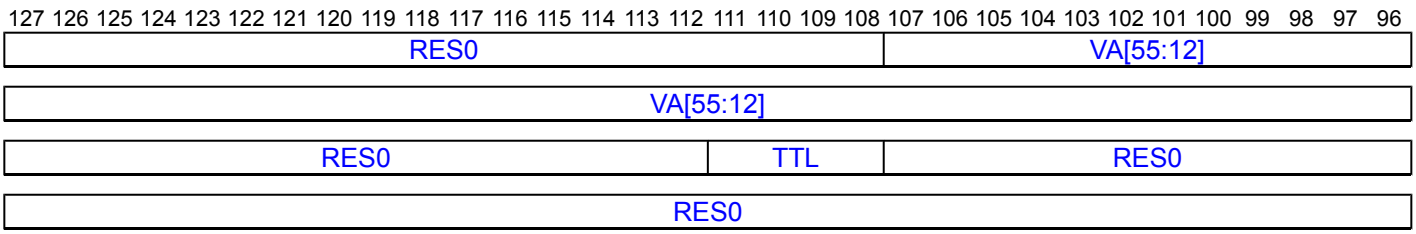
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAAE1, TLBIP VAAE1NXS are UNDEFINED.

Attributes

TLBIP VAAE1, TLBIP VAAE1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAAE1, TLBIP VAAE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAAE1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAAE1NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAAE1IS, TLBIP VAAE1ISNXS, TLB Invalidate Pair by VA, All ASID, EL1, Inner Shareable

The TLBIP VAAE1IS, TLBIP VAAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

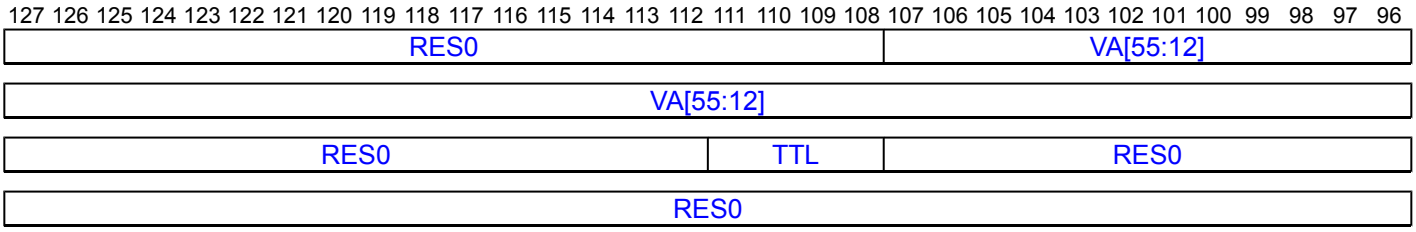
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAAE1IS, TLBIP VAAE1ISNXS are UNDEFINED.

Attributes

TLBIP VAAE1IS, TLBIP VAAE1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAAE1IS, TLBIP VAAE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAAE1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0011	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAAE1ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0011	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAAE1OS, TLBIP VAAE1OSNXS, TLB Invalidate Pair by VA, All ASID, EL1, Outer Shareable

The TLBIP VAAE1OS, TLBIP VAAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

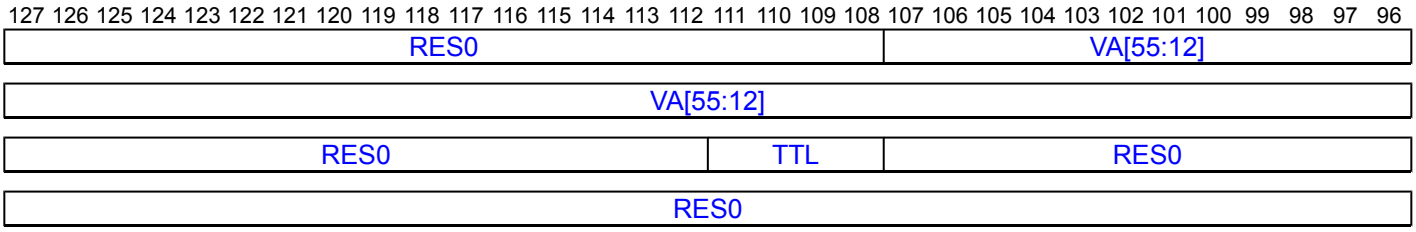
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAAE1OS, TLBIP VAAE1OSNXS are UNDEFINED.

Attributes

TLBIP VAAE1OS, TLBIP VAAE1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAAE1OS, TLBIP VAAE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAAE1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0001	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAAE1OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0001	0b011

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAALE1, TLBIP VAALE1NXS, TLB Invalidate Pair by VA, All ASID, Last level, EL1

The TLBIP VAALE1, TLBIP VAALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from the final level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

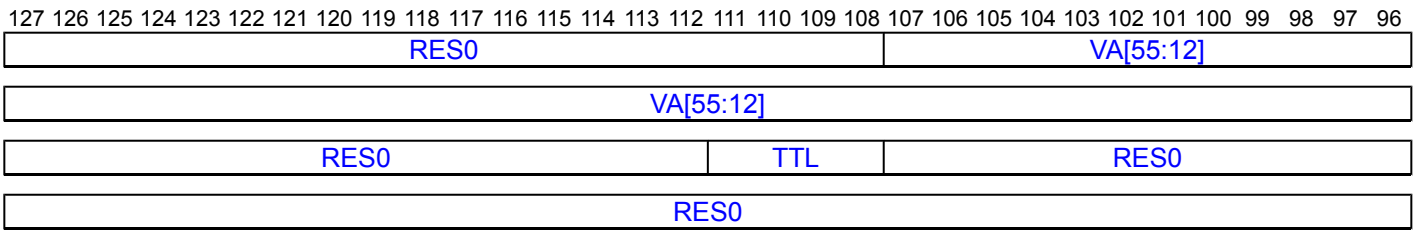
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAALE1, TLBIP VAALE1NXS are UNDEFINED.

Attributes

TLBIP VAALE1, TLBIP VAALE1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAALE1, TLBIP VAALE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAALE1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0111	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAALE1NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0111	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAALE1IS, TLBIP VAALE1ISNXS, TLB Invalidate Pair by VA, All ASID, Last Level, EL1, Inner Shareable

The TLBIP VAALE1IS, TLBIP VAALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from the final level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

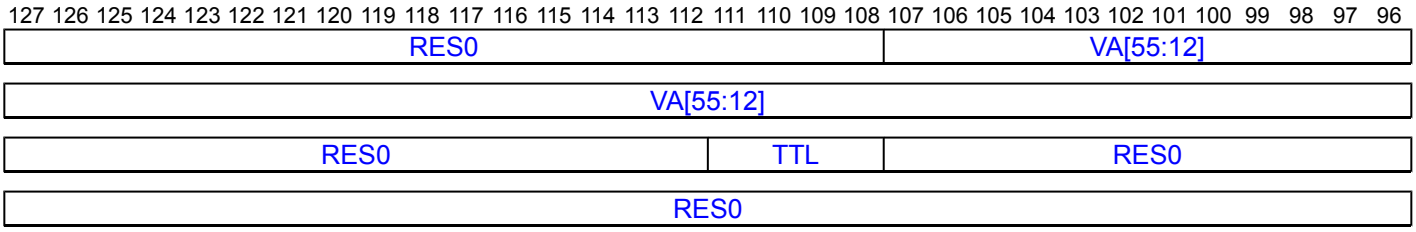
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAALE1IS, TLBIP VAALE1ISNXS are UNDEFINED.

Attributes

TLBIP VAALE1IS, TLBIP VAALE1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAALE1IS, TLBIP VAALE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAALE1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0011	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAALE1ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0011	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAALE1OS, TLBIP VAALE1OSNXS, TLB Invalidate Pair by VA, All ASID, Last Level, EL1, Outer Shareable

The TLBIP VAALE1OS, TLBIP VAALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry, from the final level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

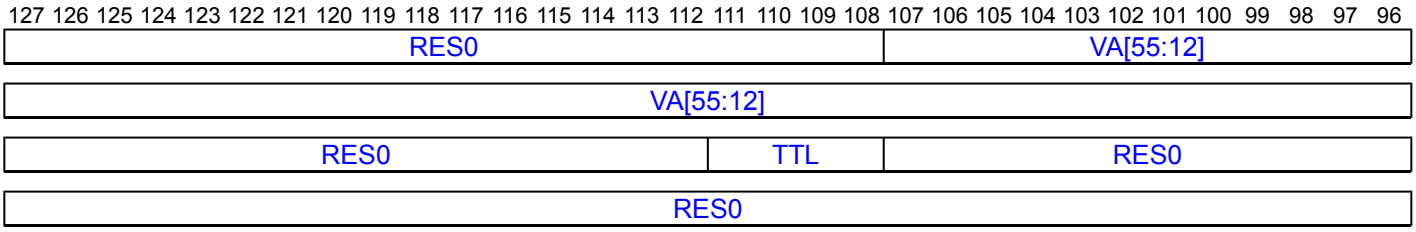
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAALE1OS, TLBIP VAALE1OSNXS are UNDEFINED.

Attributes

TLBIP VAALE1OS, TLBIP VAALE1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAALE1OS, TLBIP VAALE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAALE1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0001	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAALE1OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0001	0b111

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAE1, TLBIP VAE1NXS, TLB Invalidate Pair by VA, EL1

The TLBIP VAE1, TLBIP VAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

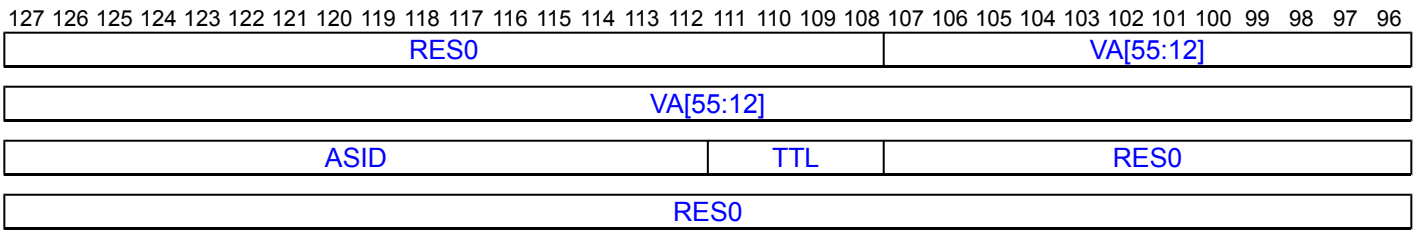
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAE1, TLBIP VAE1NXS are UNDEFINED.

Attributes

TLBIP VAE1, TLBIP VAE1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAE1, TLBIP VAE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAE1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0111	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAE1NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0111	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAE1IS, TLBIP VAE1ISNXS, TLB Invalidate Pair by VA, EL1, Inner Shareable

The TLBIP VAE1IS, TLBIP VAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

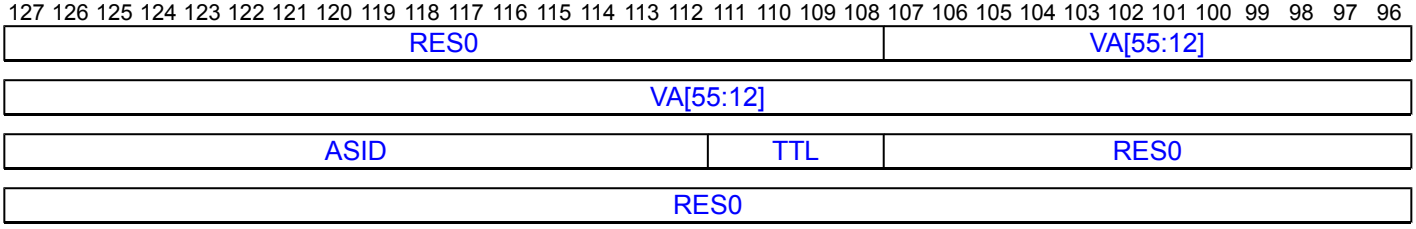
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAE1IS, TLBIP VAE1ISNXS are UNDEFINED.

Attributes

TLBIP VAE1IS, TLBIP VAE1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAE1IS, TLBIP VAE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAE1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAE1ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAE1OS, TLBIP VAE1OSNXS, TLB Invalidate Pair by VA, EL1, Outer Shareable

The TLBIP VAE1OS, TLBIP VAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

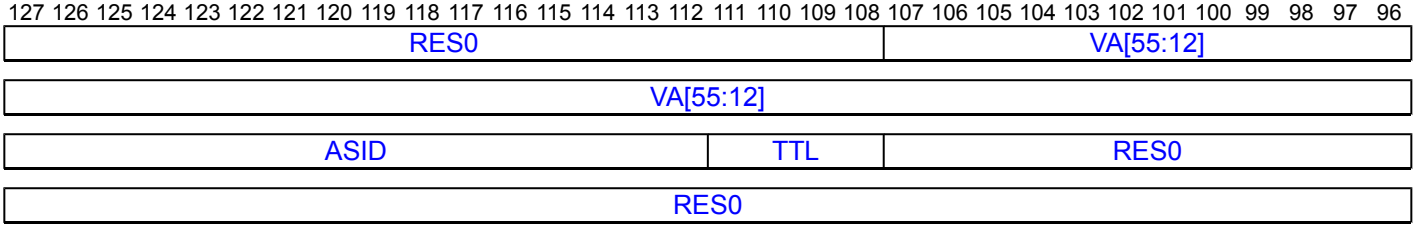
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAE1OS, TLBIP VAE1OSNXS are UNDEFINED.

Attributes

TLBIP VAE1OS, TLBIP VAE1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAE1OS, TLBIP VAE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAE1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP VAE1OSNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAE2, TLBIP VAE2NXS, TLB Invalidate Pair by VA, EL2

The TLBIP VAE2, TLBIP VAE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

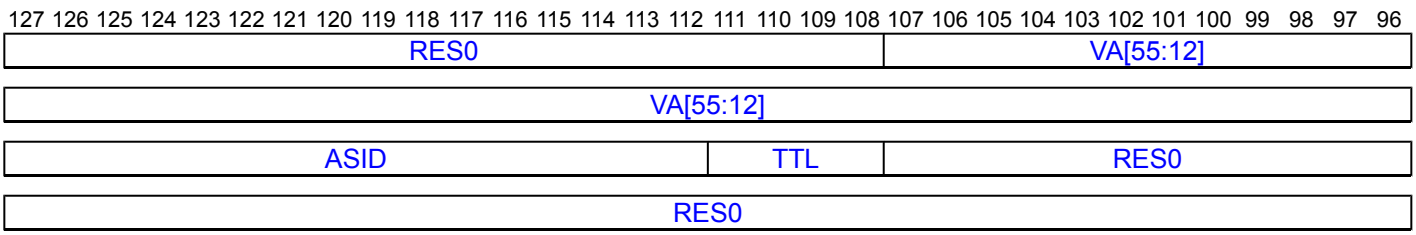
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAE2, TLBIP VAE2NXS are UNDEFINED.

Attributes

TLBIP VAE2, TLBIP VAE2NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAE2, TLBIP VAE2NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAE2{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0111	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAE2NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0111	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAE2IS, TLBIP VAE2ISNXS, TLB Invalidate Pair by VA, EL2, Inner Shareable

The TLBIP VAE2IS, TLBIP VAE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

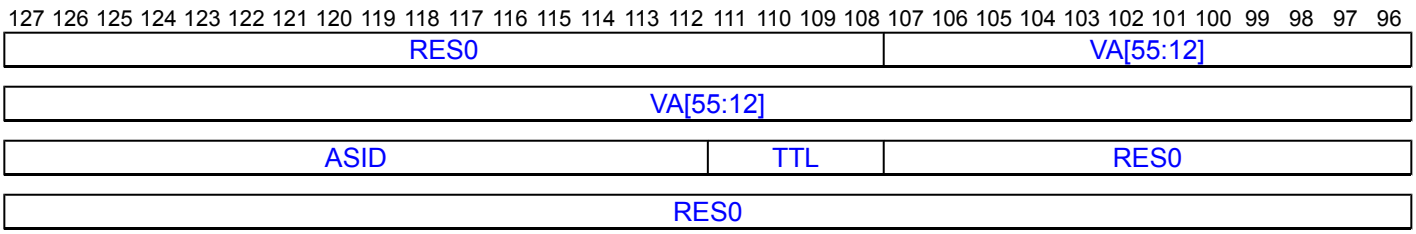
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAE2IS, TLBIP VAE2ISNXS are UNDEFINED.

Attributes

TLBIP VAE2IS, TLBIP VAE2ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAE2IS, TLBIP VAE2ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAE2IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAE2ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAE2OS, TLBIP VAE2OSNXS, TLB Invalidate Pair by VA, EL2, Outer Shareable

The TLBIP VAE2OS, TLBIP VAE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from any level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

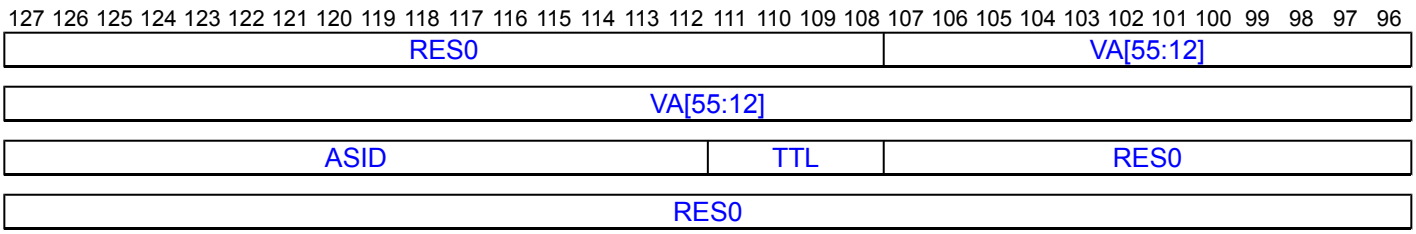
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAE2OS, TLBIP VAE2OSNXS are UNDEFINED.

Attributes

TLBIP VAE2OS, TLBIP VAE2OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAE2OS, TLBIP VAE2OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAE2OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VAE2OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAE3, TLBIP VAE3NXS, TLB Invalidate Pair by VA, EL3

The TLBIP VAE3, TLBIP VAE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

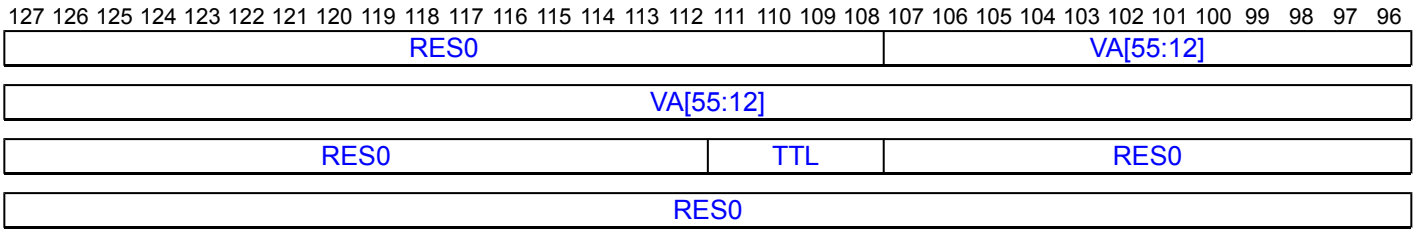
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAE3, TLBIP VAE3NXS are UNDEFINED.

Attributes

TLBIP VAE3, TLBIP VAE3NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.

- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAE3, TLBIP VAE3NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAE3{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0111	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP VAE3NXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0111	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAE3IS, TLBIP VAE3ISNXS, TLB Invalidate Pair by VA, EL3, Inner Shareable

The TLBIP VAE3IS, TLBIP VAE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

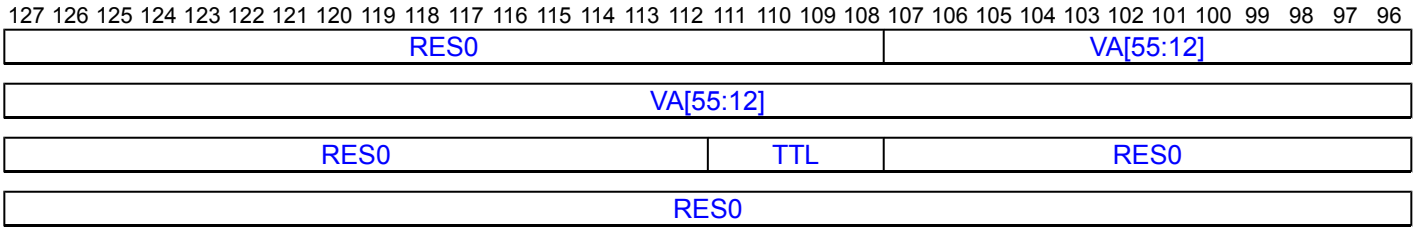
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAE3IS, TLBIP VAE3ISNXS are UNDEFINED.

Attributes

TLBIP VAE3IS, TLBIP VAE3ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAE3IS, TLBIP VAE3ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAE3IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP VAE3ISNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VAE3OS, TLBIP VAE3OSNXS, TLB Invalidate Pair by VA, EL3, Outer Shareable

The TLBIP VAE3OS, TLBIP VAE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from any level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from any level of the translation table walk, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

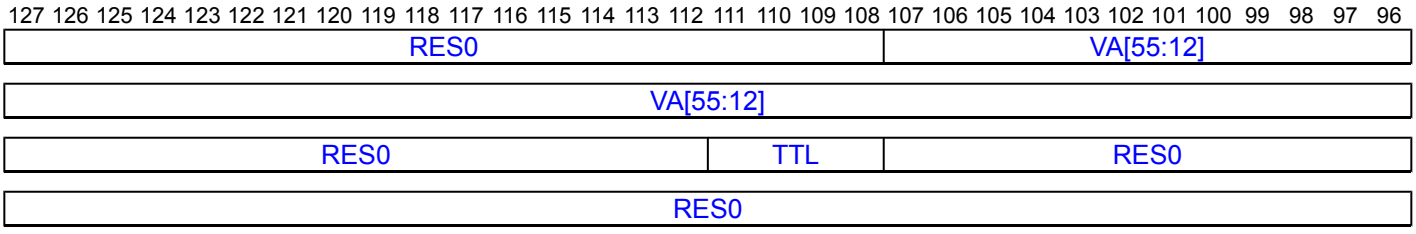
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VAE3OS, TLBIP VAE3OSNXS are UNDEFINED.

Attributes

TLBIP VAE3OS, TLBIP VAE3OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VAE3OS, TLBIP VAE3OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VAE3OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t, t2, 128]);
```

TLBIP VAE3OSNXS{, <Xt>, <Xt2>}

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VALE1, TLBIP VALE1NXS, TLB Invalidate Pair by VA, Last level, EL1

The TLBIP VALE1, TLBIP VALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

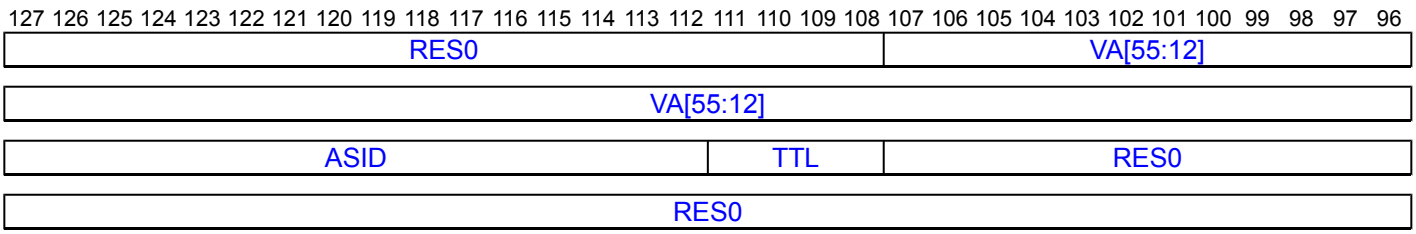
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VALE1, TLBIP VALE1NXS are UNDEFINED.

Attributes

TLBIP VALE1, TLBIP VALE1NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VALE1, TLBIP VALE1NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VALE1{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0111	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VALE1NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0111	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VALE1IS, TLBIP VALE1ISNXS, TLB Invalidate Pair by VA, Last level, EL1, Inner Shareable

The TLBIP VALE1IS, TLBIP VALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

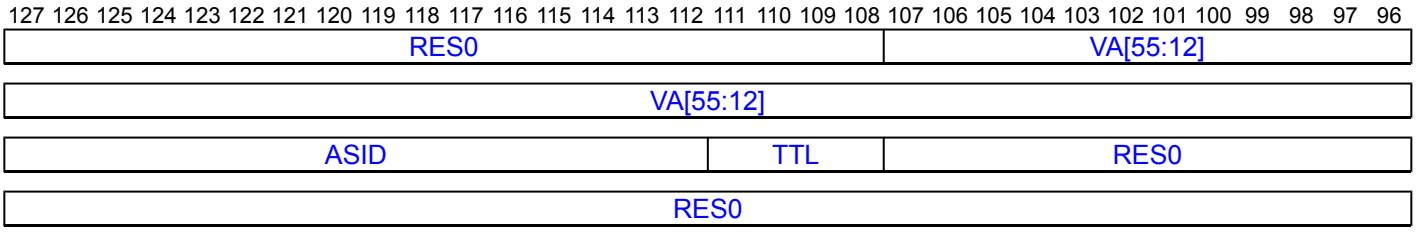
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VALE1IS, TLBIP VALE1ISNXS are UNDEFINED.

Attributes

TLBIP VALE1IS, TLBIP VALE1ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VALE1IS, TLBIP VALE1ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VALE1IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0011	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VALE1ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0011	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VALE1OS, TLBIP VALE1OSNXS, TLB Invalidate Pair by VA, Last level, EL1, Outer Shareable

The TLBIP VALE1OS, TLBIP VALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
  - If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
  - If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

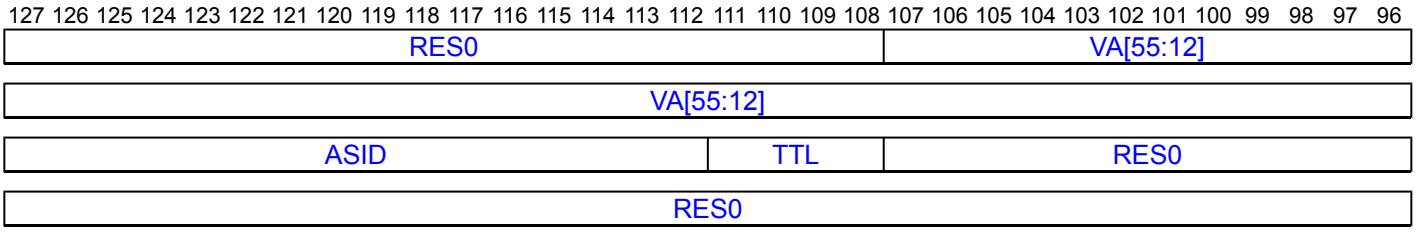
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VALE1OS, TLBIP VALE1OSNXS are UNDEFINED.

Attributes

TLBIP VALE1OS, TLBIP VALE1OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VALE1OS, TLBIP VALE1OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VALE1OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1000	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGITR_EL2.TLBIVALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VALE1OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b000	0b1001	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& IsFeatureImplemented(FEAT_HCX) && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') &&
HFGITR_EL2.TLBIVALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL0) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL0) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL1) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VALE2, TLBIP VALE2NXS, TLB Invalidate Pair by VA, Last level, EL2

The TLBIP VALE2, TLBIP VALE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

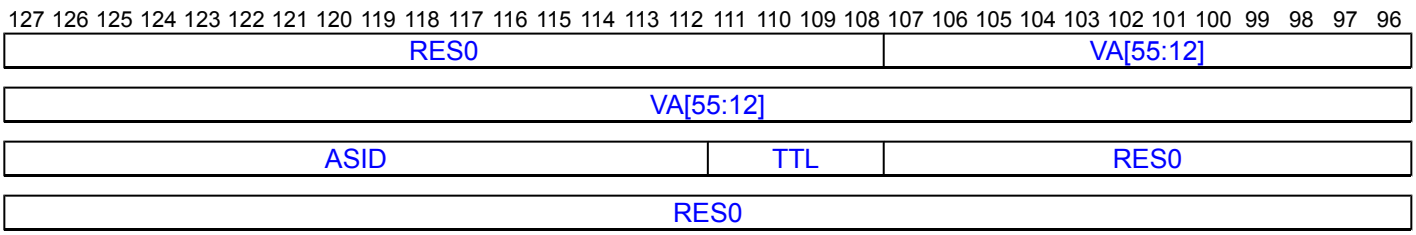
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VALE2, TLBIP VALE2NXS are UNDEFINED.

Attributes

TLBIP VALE2, TLBIP VALE2NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VALE2, TLBIP VALE2NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VALE2{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0111	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VALE2NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0111	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VALE2IS, TLBIP VALE2ISNXS, TLB Invalidate Pair by VA, Last level, EL2, Inner Shareable

The TLBIP VALE2IS, TLBIP VALE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 64-bit stage 1 translation table entry.
  - If FEAT_D128 is implemented, a 128-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

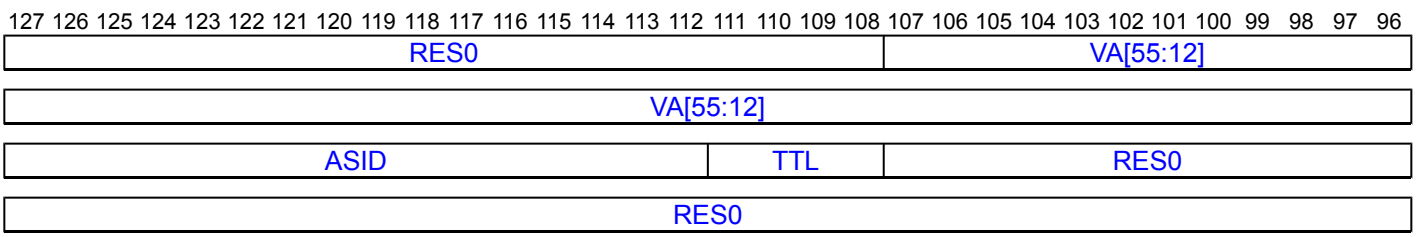
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VALE2IS, TLBIP VALE2ISNXS are UNDEFINED.

Attributes

TLBIP VALE2IS, TLBIP VALE2ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VALE2IS, TLBIP VALE2ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VALE2IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0011	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VALE2ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0011	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VALE2OS, TLBIP VALE2OSNXS, TLB Invalidate Pair by VA, Last level, EL2, Outer Shareable

The TLBIP VALE2OS, TLBIP VALE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following:
  - A 128-bit stage 1 translation table entry.
  - A 64-bit stage 1 translation table entry, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime, as determined by the Effective value of HCR_EL2.E2H, for the Security state.
- If the Effective value of HCR_EL2.E2H is not 1, the entry is from the final level of the translation table walk.
- If the Effective value of HCR_EL2.E2H is 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of SCR_EL3.NS if FEAT_RME is not implemented, or SCR_EL3.{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

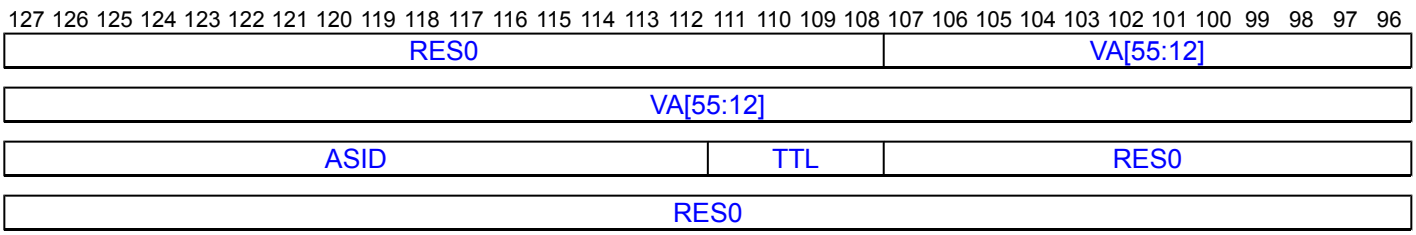
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VALE2OS, TLBIP VALE2OSNXS are UNDEFINED.

Attributes

TLBIP VALE2OS, TLBIP VALE2OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

ASID, bits [63:48]

When ELIsInHost(EL2):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VALE2OS, TLBIP VALE2OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VALE2OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1000	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VALE2OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b100	0b1001	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
    else
        if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL2) then
            return;
        else
            AArch64.TLBIP_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VALE3, TLBIP VALE3NXS, TLB Invalidate Pair by VA, Last level, EL3

The TLBIP VALE3, TLBIP VALE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from the final level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

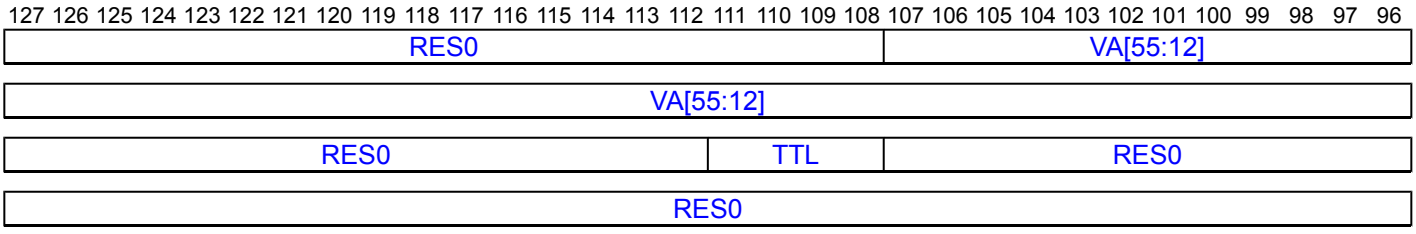
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VALE3, TLBIP VALE3NXS are UNDEFINED.

Attributes

TLBIP VALE3, TLBIP VALE3NXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VALE3, TLBIP VALE3NXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VALE3{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0111	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VALE3NXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0111	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VALE3IS, TLBIP VALE3ISNXS, TLB Invalidate Pair by VA, Last level, EL3, Inner Shareable

The TLBIP VALE3IS, TLBIP VALE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from the final level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

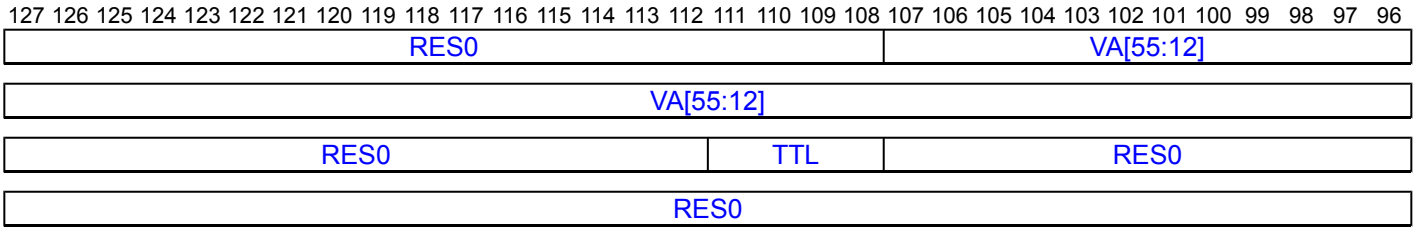
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VALE3IS, TLBIP VALE3ISNXS are UNDEFINED.

Attributes

TLBIP VALE3IS, TLBIP VALE3ISNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VALE3IS, TLBIP VALE3ISNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VALE3IS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0011	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VALE3ISNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0011	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIP VALE3OS, TLBIP VALE3OSNXS, TLB Invalidate Pair by VA, Last level, EL3, Outer Shareable

The TLBIP VALE3OS, TLBIP VALE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is one of the following
  - A 128-bit stage 1 translation table entry, from the final level of the translation table walk.
  - A 64-bit stage 1 translation table entry, from the final level of the translation table walk, if TTL[3:2] is `0b00`.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

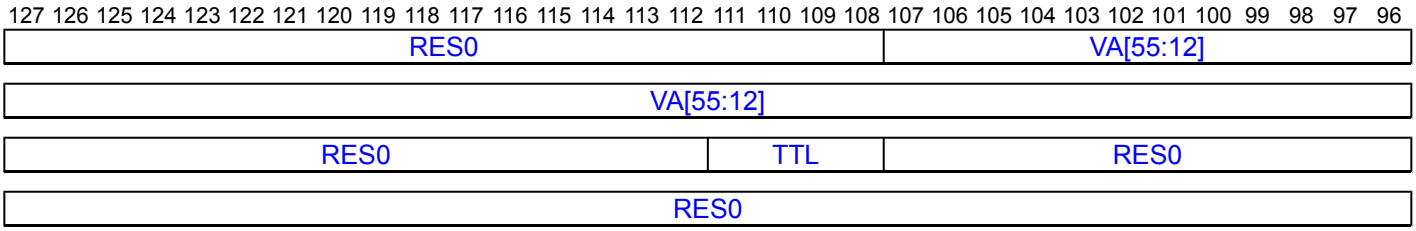
Configuration

This instruction is present only when FEAT_D128 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TLBIP VALE3OS, TLBIP VALE3OSNXS are UNDEFINED.

Attributes

TLBIP VALE3OS, TLBIP VALE3OSNXS is a 128-bit System instruction.

Field descriptions



Bits [127:108]

Reserved, RES0.

VA[55:12], bits [107:64]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

TTL	Meaning
0b00xx	No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0.
0b01xx	The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Level 0. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b10xx	The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.
0b11xx	The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:0]

Reserved, RES0.

Executing TLBIP VALE3OS, TLBIP VALE3OSNXS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBIP VALE3OS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1000	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t, t2, 128]);
```

```
TLBIP VALE3OSNXS{, <Xt>, <Xt2>}
```

op0	op1	CRn	CRm	op2
0b01	0b110	0b1001	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_D128) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RME) && !ValidSecurityStateAtEL(EL3) then
        return;
    else
        AArch64.TLBIP_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID_NONE, Broadcast_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t, t2, 128]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDR2_EL0, EL0 Read/Write Software Thread ID Register 2

The TPIDR2_EL0 characteristics are:

Purpose

Provides a location where SME-aware software executing at EL0 can store thread identifying information, for context management purposes.

The PE makes no use of this register.

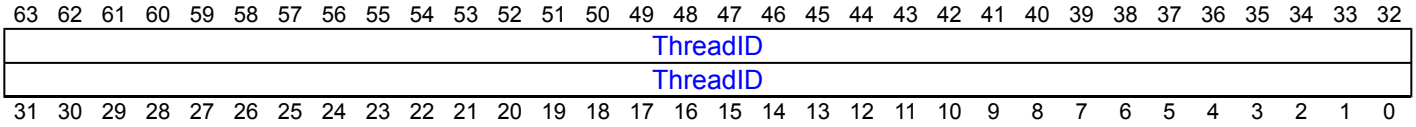
Configuration

This register is present only when FEAT_SME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDR2_EL0 are UNDEFINED.

Attributes

TPIDR2_EL0 is a 64-bit register.

Field descriptions



ThreadID, bits [63:0]

Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR2_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDR2_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.EnTP2 == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnTP2 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGRTR_EL2.nTPIDR2_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TPIDR2_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.nTPIDR2_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TPIDR2_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TPIDR2_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDR2_EL0;
```

```
MSR TPIDR2_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.EnTP2 == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnTP2 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGWTR_EL2.nTPIDR2_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR2_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.nTPIDR2_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR2_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR2_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TPIDR2_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDR_EL0, EL0 Read/Write Software Thread ID Register

The TPIDR_EL0 characteristics are:

Purpose

Provides a location where software executing at EL0 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

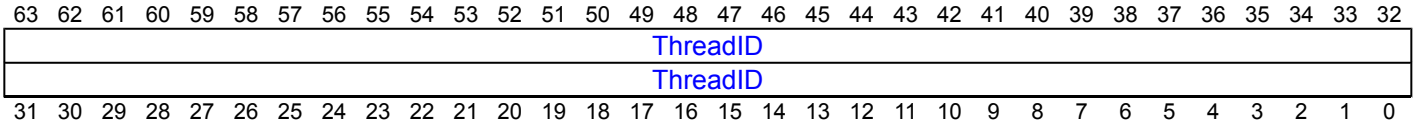
AArch64 System register TPIDR_EL0 bits [31:0] are architecturally mapped to AArch32 System register TPIDRURW[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDR_EL0 are UNDEFINED.

Attributes

TPIDR_EL0 is a 64-bit register.

Field descriptions



ThreadID, bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGRTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = TPIDR_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = TPIDR_EL0;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TPIDR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDR_EL0;
```

```
MSR TPIDR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGWTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    TPIDR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TPIDR_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDR_EL1, EL1 Software Thread ID Register

The TPIDR_EL1 characteristics are:

Purpose

Provides a location where software executing at EL1 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

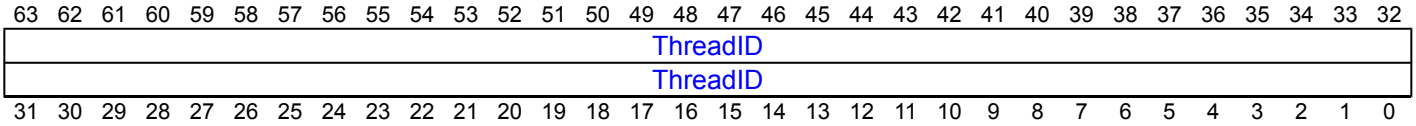
AArch64 System register TPIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TPIDRPRW[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDR_EL1 are UNDEFINED.

Attributes

TPIDR_EL1 is a 64-bit register.

Field descriptions



ThreadID, bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.TPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = TPIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TPIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDR_EL1;
```

```
MSR TPIDR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGWTR_EL2.TPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    TPIDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TPIDR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDR_EL2, EL2 Software Thread ID Register

The TPIDR_EL2 characteristics are:

Purpose

Provides a location where software executing at EL2 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch64 System register TPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HTPIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDR_EL2 are UNDEFINED.

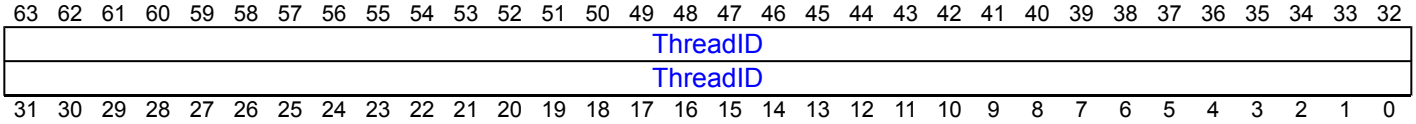
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

TPIDR_EL2 is a 64-bit register.

Field descriptions



ThreadID, bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x090];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TPIDR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDR_EL2;
```

```
MSR TPIDR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x090] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TPIDR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TPIDR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDR_EL3, EL3 Software Thread ID Register

The TPIDR_EL3 characteristics are:

Purpose

Provides a location where software executing at EL3 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

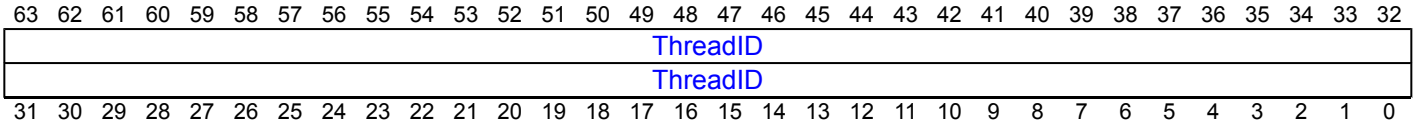
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDR_EL3 are UNDEFINED.

Attributes

TPIDR_EL3 is a 64-bit register.

Field descriptions



ThreadID, bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1101	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDR_EL3;
```

```
MSR TPIDR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1101	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.TPIDR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDRRO_EL0, EL0 Read-Only Software Thread ID Register

The TPIDRRO_EL0 characteristics are:

Purpose

Provides a location where software executing at EL1 or higher can store thread identifying information that is visible to software executing at EL0, for OS management purposes.

The PE makes no use of this register.

Configuration

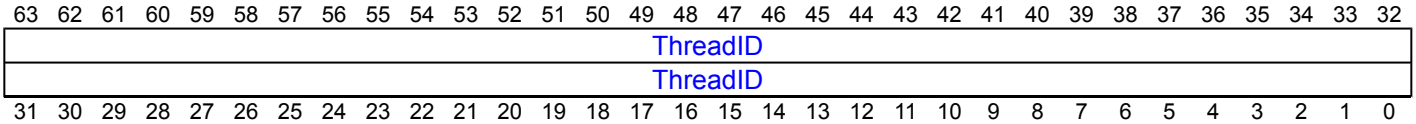
AArch64 System register TPIDRRO_EL0 bits [31:0] are architecturally mapped to AArch32 System register TPIDRURO[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDRRO_EL0 are UNDEFINED.

Attributes

TPIDRRO_EL0 is a 64-bit register.

Field descriptions



ThreadID, bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

Accessing TPIDRRO_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDRRO_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGRTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = TPIDRRO_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = TPIDRRO_EL0;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TPIDRRO_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDRRO_EL0;
```

```
MSR TPIDRRO_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGWTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDRRO_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    TPIDRRO_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TPIDRRO_EL0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBBASER_EL1, Trace Buffer Base Address Register

The TRBBASER_EL1 characteristics are:

Purpose

Defines the base address for the trace buffer.

Configuration

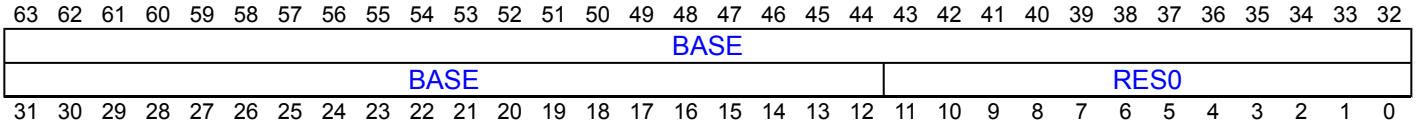
AArch64 System register TRBBASER_EL1 bits [63:0] are architecturally mapped to External register TRBBASER_EL1[63:0] when FEAT_TRBE_EXT is implemented.

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBBASER_EL1 are UNDEFINED.

Attributes

TRBBASER_EL1 is a 64-bit register.

Field descriptions



BASE, bits [63:12]

Trace Buffer Base pointer address. (TRBBASER_EL1.BASE << 12) is the address of the first byte in the trace buffer. Bits [11:0] of the Base pointer address are always zero. If the smallest implemented translation granule is not 4KB, then TRBBASER_EL1[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value Log_2(smallest implemented translation granule).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

Accessing TRBBASER_EL1

The PE might ignore a write to TRBBASER_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBBASER_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b010

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRBBASER_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBBASER_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBBASER_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBBASER_EL1;
```

```
MSR TRBBASER_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b010

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRBBASER_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBBASER_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBBASER_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBBASER_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBIDR_EL1, Trace Buffer ID Register

The TRBIDR_EL1 characteristics are:

Purpose

Describes constraints on using the Trace Buffer Unit to software, including whether the Trace Buffer Unit can be programmed at the current Exception level.

Configuration

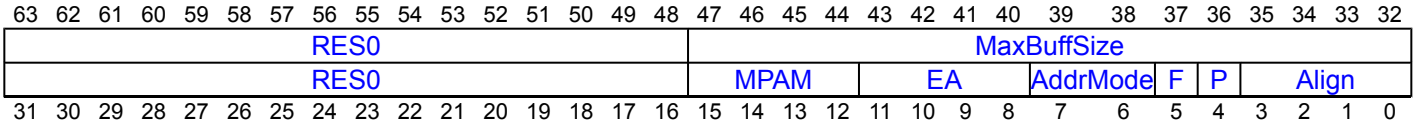
AArch64 System register TRBIDR_EL1 bits [63:0] are architecturally mapped to External register TRBIDR_EL1[63:0] when FEAT_TRBE_EXT is implemented.

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBIDR_EL1 are UNDEFINED.

Attributes

TRBIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:48]

Reserved, RES0.

MaxBuffSize, bits [47:32]

Maximum supported trace buffer size. Reserved for software use.

The only permitted value is `0x0000`, indicating there is no limit to the maximum buffer size.

Note

Permitted values relate to the values an implementation is permitted to set this field to. A hypervisor might trap accesses to this register and use other values to describe limitations of its virtualization support to a guest operating system, as follows:

- MaxBuffSize bits[8:0] encodes a mantissa value, M.
- MaxBuffSize bits[13:9] encodes an exponent value, E.
- MaxBuffSize bits[15:14] are reserved.

The maximum buffer size, in bytes, is expressed using the following function:

if IsZero(E) then UInt(M:Zeros(12)) else UInt('1':M:Zeros(UInt(E)+11))

For example:

- A value of `0x0001` means a maximum buffer size of 4KB.
- A value of `0x3FFF` means a maximum buffer size of 4092TB.

Reads as `0x0000`.

Access to this field is **RO**.

Bits [31:16]

Reserved, RES0.

MPAM, bits [15:12]

When FEAT_TRBE_EXT is implemented:

MPAM extensions. Indicates Memory Partitioning and Monitoring (MPAM) support in the Trace Buffer Unit when using External mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MPAM	Meaning
0b0000	Trace Buffer External Mode is not implemented or MPAM is not implemented by the PE.
0b0001	MPAM implemented by the Trace Buffer Unit, using default PARTID and PMG values in External mode.
0b0010	Trace Buffer MPAM extensions implemented.

When FEAT_MPAM is not implemented by the PE or FEAT_TRBE_EXT is not implemented by the PE, the only permitted value is 0b0000.

When FEAT_MPAM and FEAT_TRBE_EXT are both implemented by the PE, the value 0b0000 is not permitted.

FEAT_TRBE_MPAM implements the functionality identified by the value 0b0010.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

EA, bits [11:8]

External Abort handling. Describes how the PE manages External aborts on writes made by the Trace Buffer Unit to the trace buffer.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EA	Meaning
0b0000	Not described.
0b0001	The PE ignores External aborts on writes made by the Trace Buffer Unit.
0b0010	An External abort on a write made by the Trace Buffer Unit generates an asynchronous SError exception at the PE.

All other values are reserved.

From Armv9.3, the value 0b0000 is not permitted.

TRBIDR_EL1.EA describes only External aborts generated by the write to memory. External aborts on a translation table walk made by the Trace Buffer Unit generate trace buffer management events reported as MMU faults using TRBSR_ELx.

Access to this field is **RO**.

AddrMode, bits [7:6]

Address Modes. Describes the addressing modes available for the trace buffer.

AddrMode	Meaning
0b00	Virtual and physical address modes are supported.
0b01	Only virtual address mode is supported.
0b10	Reserved for software use under virtualization, to show that only physical address mode is supported.

Other values are reserved.

If the Effective value of TRFCR_EL2.DnVM is 1 and the value returned for TRBIDR_EL1.P is 0, then this field reads as 0b01. Otherwise, this field reads as 0b00.

Note

A hypervisor might trap accesses to this register to describe limitations of its virtualization support to a guest operating system.

F, bit [5]

Flag updates. Describes how address translations performed by the Trace Buffer Unit manage the Access flag and dirty state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F	Meaning
0b0	Hardware management of the Access flag and dirty state for accesses made by the Trace Buffer Unit is always disabled for all translation stages.
0b1	Hardware management of the Access flag and dirty state for accesses made by the Trace Buffer Unit is controlled in the same way as explicit memory accesses in the trace buffer owning translation regime.

Note

If hardware management of the Access flag is disabled for a stage of translation, an access to a Page or Block with the Access flag bit not set in the descriptor will generate an Access Flag fault.

If hardware management of the dirty state is disabled for a stage of translation, an access to a Page or Block will ignore the Dirty Bit Modifier in the descriptor and might generate a Permission fault, depending on the values of the access permission bits in the descriptor.

From Armv9.3, the value 0 is not permitted.

Access to this field is **RO**.

P, bit [4]

Programming not allowed. When read at EL3, this field reads as zero. Otherwise, indicates that the trace buffer owning Exception level is a higher Exception level or the trace buffer owning Security state is not the current Security state.

P	Meaning
0b0	Programming is allowed.
0b1	Programming not allowed.

The value read from this field depends on the current Exception level and the Effective values of MDCR_EL3.NSTB, MDCR_EL3.NSTBE, and MDCR_EL2.E2TB:

- If EL3 is implemented, MDCR_EL3.NSTB is 0b0x, and either FEAT_RME is not implemented, or Secure state is implemented and MDCR_EL3.NSTBE is 0, then this field reads as one from:
  - Non-secure EL1 and Non-secure EL2.
  - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
  - If Secure EL2 is implemented and enabled, and MDCR_EL2.E2TB is 0b00, Secure EL1.
- If EL3 is implemented, MDCR_EL3.NSTB is 0b1x and either FEAT_RME is not implemented or MDCR_EL3.NSTBE is 0, then this field reads as one from:
  - If Secure state is implemented, Secure EL1.
  - If Secure EL2 is implemented, Secure EL2.
  - If EL2 is implemented and MDCR_EL2.E2TB is 0b00, Non-secure EL1.
  - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
- If FEAT_RME is implemented, and MDCR_EL3.{NSTB, NSTBE} is {0b1x, 1}, then this field reads as one from:
  - Non-secure EL1 and Non-secure EL2.
  - If Secure state is implemented, Secure EL1 and Secure EL2.
  - If MDCR_EL2.E2TB is 0b00, Realm EL1.
- If EL3 is not implemented, EL2 is implemented, and MDCR_EL2.E2TB is 0b00, then this field reads as one from EL1.

Otherwise, this field reads as zero.

Align, bits [3:0]

Defines the minimum alignment constraint for writes to TRBPTR_EL1 and TRBTRG_EL1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Align	Meaning
0b0000	Byte.
0b0001	Halfword.
0b0010	Word.
0b0011	Doubleword.
0b0100	16 bytes.
0b0101	32 bytes.
0b0110	64 bytes.
0b0111	128 bytes.
0b1000	256 bytes.
0b1001	512 bytes.
0b1010	1KB.
0b1011	2KB.

All other values are reserved.

Access to this field is **RO**.

Accessing TRBIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b111

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.TRBIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBIDR_EL1;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBIDR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBIDR_EL1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBLIMITR_EL1, Trace Buffer Limit Address Register

The TRBLIMITR_EL1 characteristics are:

Purpose

Defines the top address for the trace buffer, and controls the trace buffer modes and enable.

Configuration

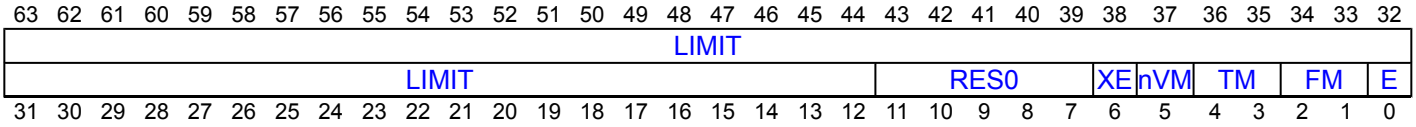
AArch64 System register TRBLIMITR_EL1 bits [63:0] are architecturally mapped to External register TRBLIMITR_EL1[63:0] when FEAT_TRBE_EXT is implemented.

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBLIMITR_EL1 are UNDEFINED.

Attributes

TRBLIMITR_EL1 is a 64-bit register.

Field descriptions



LIMIT, bits [63:12]

Trace Buffer Limit pointer address. (TRBLIMITR_EL1.LIMIT << 12) is the address of the last byte in the trace buffer plus one. Bits [11:0] of the Limit pointer address are always zero. If the smallest implemented translation granule is not 4KB, then TRBLIMITR_EL1[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value Log_2(smallest implemented translation granule).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [11:7]

Reserved, RES0.

XE, bit [6]

When FEAT_TRBE_EXT is implemented:

Trace Buffer Unit External mode enable. Used for save/restore of TRBLIMITR_EL1.XE.

XE	Meaning
0b0	Trace Buffer Unit is not enabled by this control.
0b1	If SelfHostedTraceEnabled() is FALSE, the Trace Buffer Unit is enabled.

Software must treat this field as UNK/SBZP when the OS Lock is unlocked.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- When !OSLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

nVM, bit [5]

Address mode.

nVM	Meaning
0b0	The trace buffer pointers are virtual addresses.
0b1	The trace buffer pointers are: • Physical address in the owning security state if the owning translation regime has no stage 2 translation. • Intermediate physical addresses in the owning security state if the owning translation regime has stage 2 translations.

If FEAT_TRBE_EXT is implemented and SelfHostedTraceEnabled() == FALSE, then the PE ignores the value of this field and the trace buffer pointers are always physical addresses.

If FEAT_TRBEv1p1 is implemented, SelfHostedTraceEnabled() == TRUE, and the Effective value of TRFCR_EL2.DnVM is 1, then the PE ignores the value of this field, and the trace buffer pointers are always virtual addresses.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES1** if all the following are true:
  - FEAT_TRBE_EXT is implemented.
  - !SelfHostedTraceEnabled().
- Otherwise, access to this field is **RW**.

TM, bits [4:3]

Trigger mode.

TM	Meaning
0b00	Stop on trigger. Flush then stop collection and raise maintenance interrupt on Trigger Event.
0b01	IRQ on trigger. Continue collection and raise maintenance interrupt on Trigger Event.
0b11	Ignore trigger. Continue collection and do not raise maintenance interrupt on Trigger Event.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

FM, bits [2:1]

Trace buffer mode.

FM	Meaning
0b00	Fill mode. Stop collection and raise maintenance interrupt on current write pointer wrap.
0b01	Wrap mode. Continue collection and raise maintenance interrupt on current write pointer wrap.
0b11	Circular Buffer mode. Continue collection and do not raise maintenance interrupt on current write pointer wrap.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Trace Buffer Unit enable. Controls whether the Trace Buffer Unit is enabled when SelfHostedTraceEnabled() == TRUE.

E	Meaning
0b0	Trace Buffer Unit is not enabled by this control.
0b1	If SelfHostedTraceEnabled() is TRUE, the Trace Buffer Unit is enabled.

If FEAT_TRBE_EXT is implemented and SelfHostedTraceEnabled() == FALSE, then TRBLIMITR_EL1.XE controls whether the Trace Buffer Unit is enabled.

If FEAT_TRBE_EXT is not implemented, then the Trace Buffer Unit is disabled when SelfHostedTraceEnabled() == FALSE.

All output is discarded by the Trace Buffer Unit when the Trace Buffer Unit is disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing TRBLIMITR_EL1

The PE might ignore a write to TRBLIMITR_EL1 if all the following are true:

- TRBLIMITR_EL1.E == 0b1.
- Either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- The write does not set TRBLIMITR_EL1.E to 0.

If FEAT_TRBE_EXT is implemented, the PE might ignore a write to TRBLIMITR_EL1 if all the following are true:

- TRBLIMITR_EL1.XE == 0b1.
- The Trace Buffer Unit is using External mode.
- The write does not set TRBLIMITR_EL1.XE to 0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBLIMITR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b000

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRBLIMITR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBLIMITR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBLIMITR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBLIMITR_EL1;
```

```
MSR TRBLIMITR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b000

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRBLIMITR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBLIMITR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBLIMITR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBLIMITR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBMAR_EL1, Trace Buffer Memory Attribute Register

The TRBMAR_EL1 characteristics are:

Purpose

Controls Trace Buffer Unit accesses to memory.

If the trace buffer pointers specify virtual addresses, the address properties are defined by the translation tables and this register is ignored.

Configuration

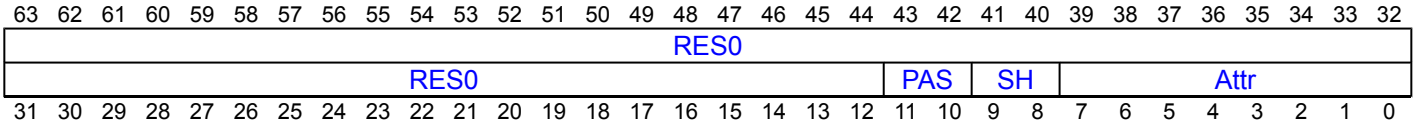
AArch64 System register TRBMAR_EL1 bits [63:0] are architecturally mapped to External register TRBMAR_EL1[63:0] when FEAT_TRBE_EXT is implemented.

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBMAR_EL1 are UNDEFINED.

Attributes

TRBMAR_EL1 is a 64-bit register.

Field descriptions



Bits [63:12]

Reserved, RES0.

PAS, bits [11:10]

When FEAT_TRBE_EXT is implemented:

Physical address specifier. Defines the PAS attribute for memory addressed by the buffer in External mode.

PAS	Meaning	Applies when
0b00	Secure.	When Secure state is implemented
0b01	Non-secure.	
0b10	Root.	When FEAT_RME is implemented
0b11	Realm.	When FEAT_RME is implemented

All other values are reserved.

If the Trace Buffer Unit is using external mode and either TRBMAR_EL1.PAS is set to a reserved value, or the IMPLEMENTATION DEFINED authentication interface prohibits invasive debug of the Security state corresponding to the physical address space selected by TRBMAR_EL1.PAS, then when the Trace Buffer Unit receives trace data from the trace unit, it does not write the trace data to memory and generates a trace buffer management event. That is, if any of the following apply:

- `ExternalInvasiveDebugEnabled()` == FALSE.
- TRBMAR_EL1.PAS is `0b00`, and either `ExternalSecureInvasiveDebugEnabled()` == FALSE or Secure state is not implemented.
- TRBMAR_EL1.PAS is `0b10`, and either `ExternalRootInvasiveDebugEnabled()` == FALSE or FEAT_RME is not implemented.
- TRBMAR_EL1.PAS is `0b11`, and either `ExternalRealmInvasiveDebugEnabled()` == FALSE or FEAT_RME is not implemented.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SH, bits [9:8]

Trace buffer shareability domain. Defines the shareability domain for Normal memory used by the trace buffer.

SH	Meaning
`0b00`	Non-shareable.
`0b10`	Outer Shareable.
`0b11`	Inner Shareable.

All other values are reserved.

This field is ignored when TRBMAR_EL1.Attr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On a Cold reset, when FEAT_TRBE_EXT is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_TRBE_EXT is not implemented, this field resets to an architecturally UNKNOWN value.

Attr, bits [7:0]

Trace buffer memory type and attributes. Defines the memory type and, for Normal memory, the cacheability attributes, for memory addressed by the trace buffer.

The encoding of this field is the same as that of a MAIR_ELx.Attr<n> field, as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of 'dd' for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise,UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise,UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Cold reset, when FEAT_TRBE_EXT is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_TRBE_EXT is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing TRBMAR_EL1

The PE might ignore a write to TRBMAR_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBMAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b100

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRBMAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMAR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMAR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMAR_EL1;
```

```
MSR TRBMAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b100

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRBMAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMAR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBMPAM_EL1, Trace Buffer MPAM Configuration Register

The TRBMPAM_EL1 characteristics are:

Purpose

Defines the PARTID, PMG, and MPAM_SP values used by the trace buffer unit in external mode.

Configuration

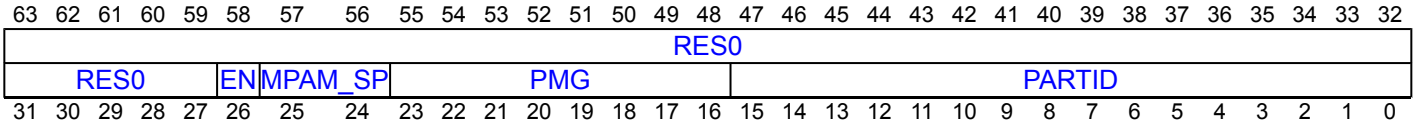
AArch64 System register TRBMPAM_EL1 bits [63:0] are architecturally mapped to External register TRBMPAM_EL1[63:0].

This register is present only when FEAT_TRBE_MPAM is implemented. Otherwise, direct accesses to TRBMPAM_EL1 are UNDEFINED.

Attributes

TRBMPAM_EL1 is a 64-bit register.

Field descriptions



Bits [63:27]

Reserved, RES0.

EN, bit [26]

Enable. Enables use of non-default MPAM values.

EN	Meaning
0b0	Use default MPAM values.
0b1	Use TRBMPAM_EL1.{PARTID, PMG, MPAM_SP}.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

MPAM_SP, bits [25:24]

Partition Identifier space. Selects the PARTID space.

MPAM_SP	Meaning	Applies when
0b00	PARTID is in the Secure PARTID space.	When Secure state is implemented
0b01	PARTID is in the Non-secure PARTID space.	
0b10	PARTID is in the Root PARTID space.	When FEAT_RME is implemented
0b11	PARTID is in the Realm PARTID space.	When FEAT_RME is implemented

If the Trace Buffer Unit is using external mode and either TRBMPAM_EL1.MPAM_SP is set to reserved value, or the IMPLEMENTATION DEFINED authentication interface prohibits invasive debug of the Security state corresponding to the Partition Identifier space selected by TRBMPAM_EL1.MPAM_SP, then when the Trace Buffer Unit receives trace data from the trace unit, it does not write the trace data to memory and generates a trace buffer management event.

The interface prohibits invasive debug of the Security state if any of the following apply:

- `ExternalInvasiveDebugEnabled()` == FALSE.
- Secure state is implemented, `ExternalSecureInvasiveDebugEnabled()` == FALSE and TRBMPAM_EL1.MPAM_SP is `0b00`.
- FEAT_RME is implemented, `ExternalRootInvasiveDebugEnabled()` == FALSE, and TRBMPAM_EL1.MPAM_SP is `0b10`.
- FEAT_RME is implemented, `ExternalRealmInvasiveDebugEnabled()` == FALSE, and TRBMPAM_EL1.MPAM_SP is `0b11`.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PMG, bits [23:16]

Performance Monitoring Group. Selects the PMG.

Only sufficient low-order bits are required to represent the TRBDEVID1.PMG_MAX. Higher-order bits are RES0.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PARTID, bits [15:0]

Partition Identifier. Selects the PARTID.

Only sufficient low-order bits are required to represent the TRBDEVID1.PARTID_MAX. Higher-order bits are RES0.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBMPAM_EL1

The PE might ignore a write to TRBMPAM_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == `0b1`, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == `0b1`, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBMPAM_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b101

```
if !IsFeatureImplemented(FEAT_TRBE_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnTB2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nTRBMPAM_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnTB2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMPAM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnTB2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnTB2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMPAM_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMPAM_EL1;
```

```
MSR TRBMPAM_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b101

```
if !IsFeatureImplemented(FEAT_TRBE_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnTB2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nTRBMPAM_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnTB2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMPAM_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnTB2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnTB2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMPAM_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMPAM_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPTR_EL1, Trace Buffer Write Pointer Register

The TRBPTR_EL1 characteristics are:

Purpose

Defines the current write pointer for the trace buffer.

Configuration

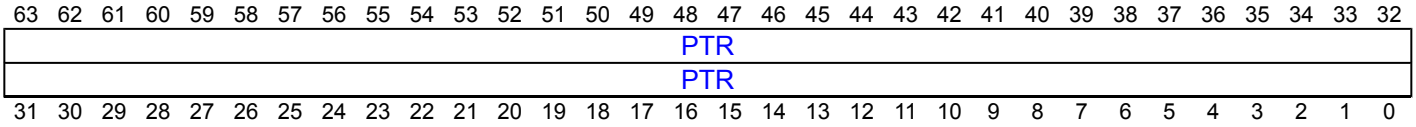
AArch64 System register TRBPTR_EL1 bits [63:0] are architecturally mapped to External register TRBPTR_EL1[63:0] when FEAT_TRBE_EXT is implemented.

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBPTR_EL1 are UNDEFINED.

Attributes

TRBPTR_EL1 is a 64-bit register.

Field descriptions



PTR, bits [63:0]

Trace Buffer current write pointer address.

Defines the virtual address of the next entry to be written to the trace buffer.

If TRBIDR_EL1.Align is not zero, then it is IMPLEMENTATION DEFINED whether bits [M-1:0] are RES0 or read/write, where M is an integer between 1 and TRBIDR_EL1.Align inclusive.

The architecture places restrictions on the values that software can write to the pointer. For more information, see 'Restrictions on Programming the Trace Buffer Unit'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBPTR_EL1

The PE might ignore a write to TRBPTR_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == `0b1`, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == `0b1`, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBPTR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b001

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRBPTR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBPTR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBPTR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBPTR_EL1;
```

```
MSR TRBPTR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b001

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRBPTR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBPTR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBPTR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBPTR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBSR_EL1, Trace Buffer Status/syndrome Register (EL1)

The TRBSR_EL1 characteristics are:

Purpose

Provides syndrome information to software for a trace buffer management event.

Configuration

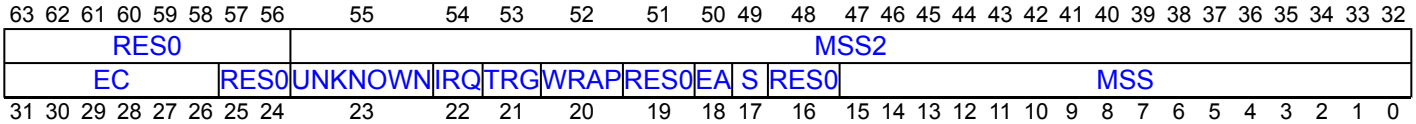
AArch64 System register TRBSR_EL1 bits [63:0] are architecturally mapped to External register TRBSR_EL1[63:0].

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBSR_EL1 are UNDEFINED.

Attributes

TRBSR_EL1 is a 64-bit register.

Field descriptions



Bits [63:56]

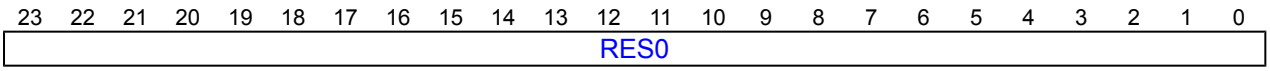
Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

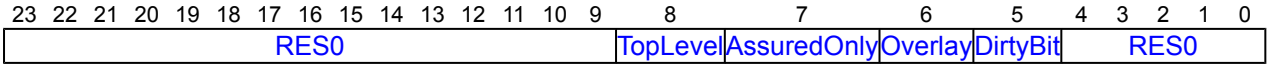
MSS2 encoding for other trace buffer management events



Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer



Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, TRBSR_EL1.EC == 0b100101, and GetTRBSR_EL1_FSC() IN {0b0011xx}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetTRBSR_EL1_FSC() IN {0b0011xx}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetTRBSR_EL1_FSC() IN {0b0011xx}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

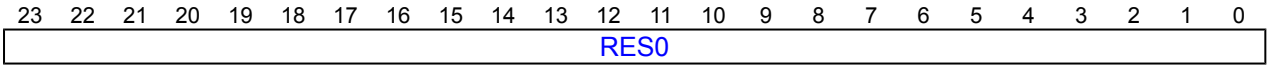
Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

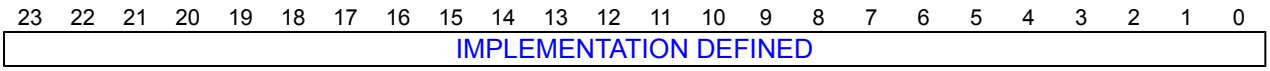
MSS2 encoding for Granule Protection Check faults on write to trace buffer



Bits [23:0]

Reserved, RES0.

MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason



IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the trace buffer management event.

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other trace buffer management event. All trace buffer management events other than those described by the other defined Event class codes.	MSS encoding for other trace buffer management events	MSS2 encoding for other trace buffer management events	
0b011110	Granule Protection Check fault on write to trace buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to trace buffer	MSS2 encoding for Granule Protection Check faults on write to trace buffer	When FEAT_RME is implemented
0b011111	Trace buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	
0b100101	Stage 2 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [25:24]

Reserved, RES0.

Bit [23]

When FEAT_TRBE_EXT is implemented:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

IRQ, bit [22]

Maintenance status. Indicates that a trace buffer management event has been recorded.

IRQ	Meaning
0b0	No trace buffer management event for EL1 has been recorded.
0b1	A trace buffer management event for EL1 has been recorded.

When FEAT_TRBE_EXC is implemented, this field indicates a management event for EL1.

If FEAT_TRBE_EXC is implemented and the TRBE Profiling exception for EL1 is enabled, then when this field is 1, a TRBE Profiling exception for EL1 is pending

If FEAT_TRBE_EXC is not implemented or the TRBE Profiling exception for EL1 is disabled, then this field drives the **TRBIRQ** trace buffer interrupt request signal.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

TRG, bit [21]

Triggered.

TRG	Meaning
0b0	No Detected Trigger has been observed since this field was last cleared to zero.
0b1	A Detected Trigger has been observed since this field was last cleared to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

WRAP, bit [20]

Wrapped.

WRAP	Meaning
0b0	The current write pointer has not wrapped since this field was last cleared to zero.
0b1	The current write pointer has wrapped since this field was last cleared to zero.

For each byte of trace the Trace Buffer Unit Accepts and writes to the trace buffer at the address in the current write pointer, if the current write pointer is equal to the Limit pointer minus one, the current write pointer is wrapped by setting it to the Base pointer, and this field is set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

EA, bit [18]

From Armv9.3:

Reserved, RES0.

When the PE sets this bit as the result of an External abort:

External Abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Trace Buffer Unit.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S, bit [17]

Stopped.

S	Meaning
0b0	Collection has not been stopped.
0b1	Collection is stopped.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [16]

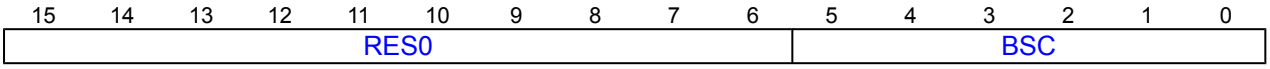
Reserved, RES0.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the trace buffer management event.

The syndrome contents for each trace buffer management event are described in the following sections.

MSS encoding for other trace buffer management events



Bits [15:6]

Reserved, RES0.

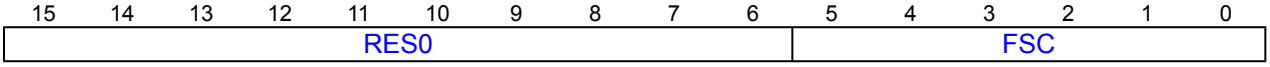
BSC, bits [5:0]

Trace buffer status code

BSC	Meaning	Applies when
0b000000	Collection not stopped, or access not allowed.	
0b000001	Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode.	
0b000010	Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information.	
0b000011	Manual Stop. Collection stopped because of a Manual Stop event. See TRBCR.ManStop for more information.	When FEAT_TRBE_EXT is implemented
0b000100	Buffer size. The requested trace buffer size was too large.	

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer



Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented

0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to trace buffer

Bits [15:0]

Reserved, RES0.

MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing TRBSR_EL1

The PE might ignore a write to TRBSR_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TRBSR_EL1 or TRBSR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} && (EffectiveTRFCR_EL2_EE() != '00' && TRFCR_EL1.EE !=
'00') then
        X[t, 64] = NVMem[0x860];
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    elsif EffectiveTRFCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        X[t, 64] = TRBSR_EL2;
    else
        X[t, 64] = TRBSR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBSR_EL1;
```

```
MSR TRBSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} && (EffectiveTRFCR_EL2_EE() != '00' && TRFCR_EL1.EE !=
'00') then
        NVMem[0x860] = X[t, 64];
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    elsif EffectiveTRFCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        TRBSR_EL2 = X[t, 64];
    else
        TRBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBSR_EL1 = X[t, 64];
```

When FEAT_TRBE_EXC is implemented and FEAT_VHE is implemented

```
MRS <Xt>, TRBSR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_TRBE_EXC) && IsFeatureImplemented(FEAT_VHE)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x860];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            X[t, 64] = TRBSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            X[t, 64] = TRBSR_EL1;
    else
        UNDEFINED;
```

When FEAT_TRBE_EXC is implemented and FEAT_VHE is implemented

```
MSR TRBSR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_TRBE_EXC) && IsFeatureImplemented(FEAT_VHE)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x860] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            TRBSR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            TRBSR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBSR_EL2, Trace Buffer Syndrome Register (EL2)

The TRBSR_EL2 characteristics are:

Purpose

Provides syndrome information to software for a trace buffer management event.

Configuration

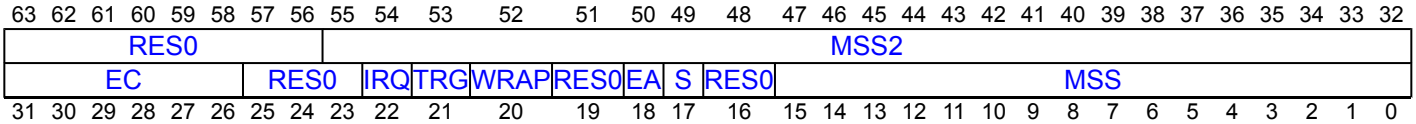
This register is present only when FEAT_TRBE_EXC is implemented. Otherwise, direct accesses to TRBSR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

TRBSR_EL2 is a 64-bit register.

Field descriptions



Bits [63:56]

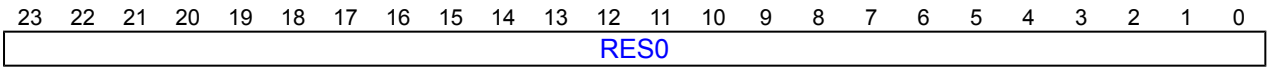
Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

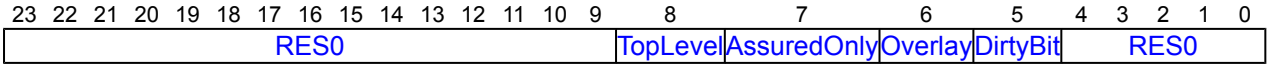
MSS2 encoding for other trace buffer management events



Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer



Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, TRBSR_EL2.EC == 0b100101, and GetTRBSR_EL2_FSC() IN {0b0011xx}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetTRBSR_EL2_FSC() IN {0b0011xx}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetTRBSR_EL2_FSC() IN {0b0011xx}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

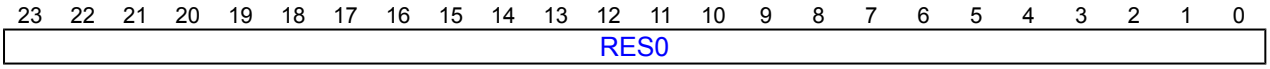
Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

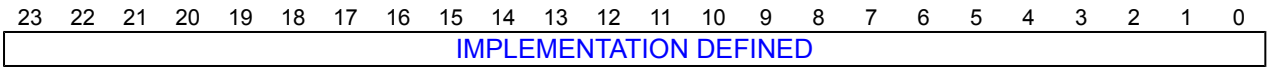
MSS2 encoding for Granule Protection Check faults on write to trace buffer



Bits [23:0]

Reserved, RES0.

MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason



IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the trace buffer management event.

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other trace buffer management event. All trace buffer management events other than those described by the other defined Event class codes.	MSS encoding for other trace buffer management events	MSS2 encoding for other trace buffer management events	
0b011110	Granule Protection Check fault on write to trace buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to trace buffer	MSS2 encoding for Granule Protection Check faults on write to trace buffer	When FEAT_RME is implemented
0b011111	Trace buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	
0b100101	Stage 2 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [25:23]

Reserved, RES0.

IRQ, bit [22]

Maintenance status. Indicates that a trace buffer management event has been recorded.

IRQ	Meaning
0b0	No trace buffer management event for EL2 has been recorded.
0b1	A trace buffer management event for EL2 has been recorded.

When FEAT_TRBE_EXC is implemented, this field indicates a management event for EL2.

If the TRBE Profiling exception for EL2 is enabled, then when this field is 1, a TRBE Profiling exception for EL2 is pending

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

TRG, bit [21]

Triggered.

TRG	Meaning
0b0	No Detected Trigger has been observed since this field was last cleared to zero.
0b1	A Detected Trigger has been observed since this field was last cleared to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

WRAP, bit [20]

Wrapped.

WRAP	Meaning
0b0	The current write pointer has not wrapped since this field was last cleared to zero.
0b1	The current write pointer has wrapped since this field was last cleared to zero.

For each byte of trace the Trace Buffer Unit Accepts and writes to the trace buffer at the address in the current write pointer, if the current write pointer is equal to the Limit pointer minus one, the current write pointer is wrapped by setting it to the Base pointer, and this field is set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

EA, bit [18]

From Armv9.3:

Reserved, RES0.

When the PE sets this bit as the result of an External abort:

External Abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Trace Buffer Unit.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S, bit [17]

Stopped.

S	Meaning
0b0	Collection has not been stopped.
0b1	Collection is stopped.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [16]

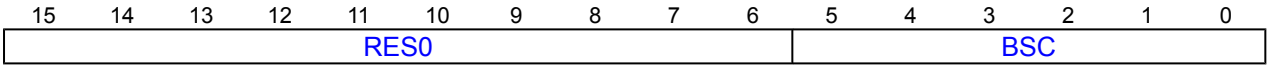
Reserved, RES0.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the trace buffer management event.

The syndrome contents for each trace buffer management event are described in the following sections.

MSS encoding for other trace buffer management events



Bits [15:6]

Reserved, RES0.

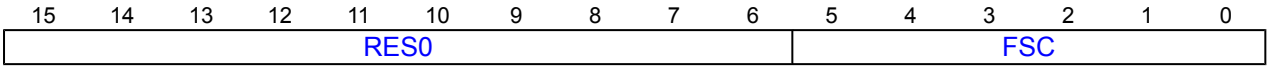
BSC, bits [5:0]

Trace buffer status code

BSC	Meaning	Applies when
0b000000	Collection not stopped, or access not allowed.	
0b000001	Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode.	
0b000010	Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information.	
0b000011	Manual Stop. Collection stopped because of a Manual Stop event. See TRBCR.ManStop for more information.	When FEAT_TRBE_EXT is implemented
0b000100	Buffer size. The requested trace buffer size was too large.	

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer



Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented

0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to trace buffer

Bits [15:0]

Reserved, RES0.

MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing TRBSR_EL2

The PE might ignore a write to TRBSR_EL2 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TRBSR_EL2 or TRBSR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBSR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE_EXC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TRBEE == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TRBEE == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TRBSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRBSR_EL2;
```

```
MSR TRBSR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE_EXC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TRBEE == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TRBEE == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRBSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRBSR_EL2 = X[t, 64];
```

MRS <Xt>, TRBSR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} && (EffectiveTRFCR_EL2_EE() != '00' && TRFCR_EL1.EE !=
'00') then
        X[t, 64] = NVMem[0x860];
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    elsif EffectiveTRFCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        X[t, 64] = TRBSR_EL2;
    else
        X[t, 64] = TRBSR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBSR_EL1;
```

MSR TRBSR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} && (EffectiveTRFCR_EL2_EE() != '00' && TRFCR_EL1.EE !=
'00') then
        NVMem[0x860] = X[t, 64];
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    elsif EffectiveTRFCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        TRBSR_EL2 = X[t, 64];
    else
        TRBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBSR_EL1 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBSR_EL3, Trace Buffer Syndrome Register (EL3)

The TRBSR_EL3 characteristics are:

Purpose

Provides syndrome information to software for a trace buffer management event.

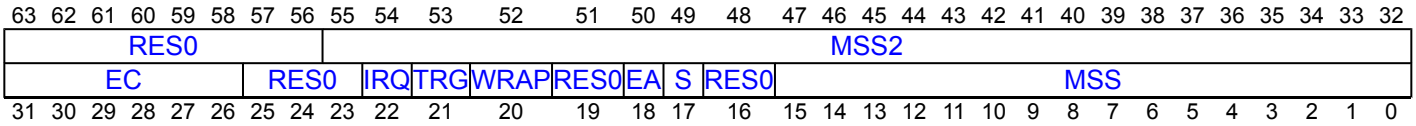
Configuration

This register is present only when FEAT_TRBE_EXC is implemented and EL3 is implemented. Otherwise, direct accesses to TRBSR_EL3 are UNDEFINED.

Attributes

TRBSR_EL3 is a 64-bit register.

Field descriptions



Bits [63:56]

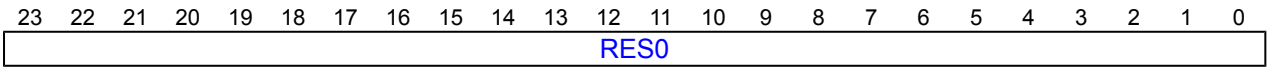
Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

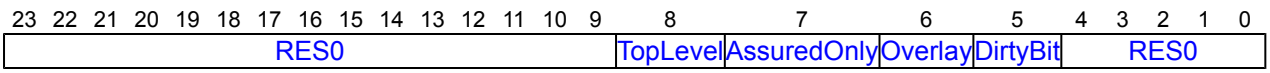
MSS2 encoding for other trace buffer management events



Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer



Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, TRBSR_EL3.EC == 0b100101, and GetTRBSR_EL3_FSC() IN {0b0011xx}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetTRBSR_EL3_FSC() IN {0b0011xx}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetTRBSR_EL3_FSC() IN {0b0011xx}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

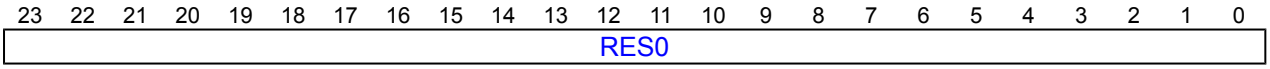
Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

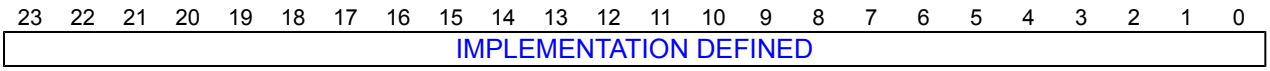
MSS2 encoding for Granule Protection Check faults on write to trace buffer



Bits [23:0]

Reserved, RES0.

MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason



IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the trace buffer management event.

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other trace buffer management event. All trace buffer management events other than those described by the other defined Event class codes.	MSS encoding for other trace buffer management events	MSS2 encoding for other trace buffer management events	
0b011110	Granule Protection Check fault on write to trace buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to trace buffer	MSS2 encoding for Granule Protection Check faults on write to trace buffer	When FEAT_RME is implemented
0b011111	Trace buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	
0b100101	Stage 2 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [25:23]

Reserved, RES0.

IRQ, bit [22]

Maintenance status. Indicates that a trace buffer management event has been recorded.

IRQ	Meaning
0b0	No trace buffer management event for EL3 has been recorded.
0b1	A trace buffer management event for EL3 has been recorded.

When FEAT_TRBE_EXC is implemented, this field indicates a management event for EL3.

If the TRBE Profiling exception for EL3 is enabled, then when this field is 1, a TRBE Profiling exception for EL3 is pending

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

TRG, bit [21]

Triggered.

TRG	Meaning
0b0	No Detected Trigger has been observed since this field was last cleared to zero.
0b1	A Detected Trigger has been observed since this field was last cleared to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

WRAP, bit [20]

Wrapped.

WRAP	Meaning
0b0	The current write pointer has not wrapped since this field was last cleared to zero.
0b1	The current write pointer has wrapped since this field was last cleared to zero.

For each byte of trace the Trace Buffer Unit Accepts and writes to the trace buffer at the address in the current write pointer, if the current write pointer is equal to the Limit pointer minus one, the current write pointer is wrapped by setting it to the Base pointer, and this field is set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

EA, bit [18]

From Armv9.3:

Reserved, RES0.

When the PE sets this bit as the result of an External abort:

External Abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Trace Buffer Unit.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S, bit [17]

Stopped.

S	Meaning
0b0	Collection has not been stopped.
0b1	Collection is stopped.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [16]

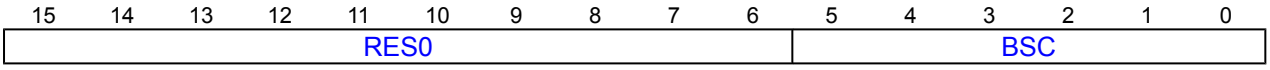
Reserved, RES0.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the trace buffer management event.

The syndrome contents for each trace buffer management event are described in the following sections.

MSS encoding for other trace buffer management events



Bits [15:6]

Reserved, RES0.

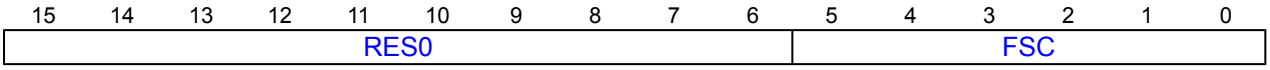
BSC, bits [5:0]

Trace buffer status code

BSC	Meaning	Applies when
0b000000	Collection not stopped, or access not allowed.	
0b000001	Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode.	
0b000010	Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information.	
0b000011	Manual Stop. Collection stopped because of a Manual Stop event. See TRBCR.ManStop for more information.	When FEAT_TRBE_EXT is implemented
0b000100	Buffer size. The requested trace buffer size was too large.	

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer



Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented

0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to trace buffer

Bits [15:0]

Reserved, RES0.

MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing TRBSR_EL3

The PE might ignore a write to TRBSR_EL3 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBSR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1001	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_TRBE_EXC) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRBSR_EL3;
```

```
MSR TRBSR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1001	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_TRBE_EXC) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TRBSR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBTRG_EL1, Trace Buffer Trigger Counter Register

The TRBTRG_EL1 characteristics are:

Purpose

Specifies the number of bytes of trace to capture following a Detected Trigger before a Trigger Event.

Configuration

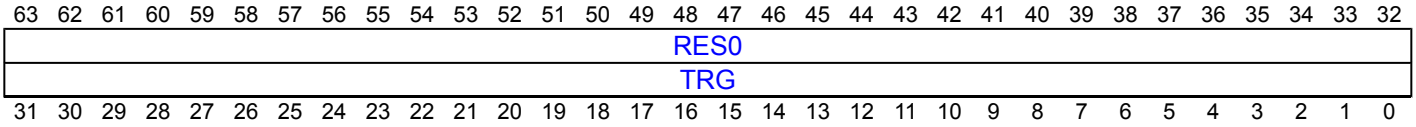
AArch64 System register TRBTRG_EL1 bits [63:0] are architecturally mapped to External register TRBTRG_EL1[63:0] when FEAT_TRBE_EXT is implemented.

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBTRG_EL1 are UNDEFINED.

Attributes

TRBTRG_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TRG, bits [31:0]

Trigger count.

Specifies the number of bytes of trace to capture following a Detected Trigger before a Trigger Event.

TRBTRG_EL1 decrements by 1 for every byte of trace written to the trace buffer when all of the following are true:

- TRBTRG_EL1 is nonzero.
- TRBSR_EL1.TRG is 1.

The architecture places restrictions on the values that software can write to the counter.

Note

As a result of the restrictions an implementation might treat some of TRG[M:0] as RES0, where M is defined by TRBIDR_EL1.Align.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBTRG_EL1

The PE might ignore a write to TRBTRG_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRBTRG_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b110

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRBTRG_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBTRG_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBTRG_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBTRG_EL1;
```

MSR TRBTRG_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b110

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRBTRG_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBTRG_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] !=
SCR_EL3.NS || (IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBTRG_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBTRG_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCACATR<n>, Trace Address Comparator Access Type Register <n>, n = 0 - 15

The TRCACATR<n> characteristics are:

Purpose

Defines the type of access for the corresponding TRCACVR<n> Register. This register configures the context type, Exception levels, alignment, masking that is applied by the Address Comparator, and how the Address Comparator behaves when it is one half of an Address Range Comparator.

Configuration

AArch64 System register TRCACATR<n> bits [63:0] are architecturally mapped to External register TRCACATR<n>[63:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMACPAIRS) * 2 > n. Otherwise, direct accesses to TRCACATR<n> are UNDEFINED.

Attributes

TRCACATR<n> is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45
RES0													EXLEVEL_RL_EL2	EXLEVEL_RL_EL1	EXLEVEL_RL_EL0	RES0	EXLEVEL_NS_EL2	EXLEVEL_NS_EL
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13

Bits [63:19]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [18]

When FEAT_RME is implemented:

Realm EL2 address comparison control. Controls whether a comparison can occur at EL2 in Realm state.

EXLEVEL_RL_EL2	Meaning
0b0	When TRCACATR<n>.EXLEVEL_NS_EL2 is 0 the Address Comparator performs comparisons in Realm EL2. When TRCACATR<n>.EXLEVEL_NS_EL2 is 1 the Address Comparator does not perform comparisons in Realm EL2.
0b1	When TRCACATR<n>.EXLEVEL_NS_EL2 is 0 the Address Comparator does not perform comparisons in Realm EL2. When TRCACATR<n>.EXLEVEL_NS_EL2 is 1 the Address Comparator performs comparisons in Realm EL2.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL1, bit [17]

When FEAT_RME is implemented:

Realm EL1 address comparison control. Controls whether a comparison can occur at EL1 in Realm state.

EXLEVEL_RL_EL1	Meaning
0b0	When TRCACATR<n>.EXLEVEL_NS_EL1 is 0 the Address Comparator performs comparisons in Realm EL1. When TRCACATR<n>.EXLEVEL_NS_EL1 is 1 the Address Comparator does not perform comparisons in Realm EL1.
0b1	When TRCACATR<n>.EXLEVEL_NS_EL1 is 0 the Address Comparator does not perform comparisons in Realm EL1. When TRCACATR<n>.EXLEVEL_NS_EL1 is 1 the Address Comparator performs comparisons in Realm EL1.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL0, bit [16]

When FEAT_RME is implemented:

Realm EL0 address comparison control. Controls whether a comparison can occur at EL0 in Realm state.

EXLEVEL_RL_EL0	Meaning
0b0	When TRCACATR<n>.EXLEVEL_NS_EL0 is 0 the Address Comparator performs comparisons in Realm EL0. When TRCACATR<n>.EXLEVEL_NS_EL0 is 1 the Address Comparator does not perform comparisons in Realm EL0.
0b1	When TRCACATR<n>.EXLEVEL_NS_EL0 is 0 the Address Comparator does not perform comparisons in Realm EL0. When TRCACATR<n>.EXLEVEL_NS_EL0 is 1 the Address Comparator performs comparisons in Realm EL0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [15]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [14]

When Non-secure EL2 is implemented:

Non-secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Non-secure state.

EXLEVEL_NS_EL2	Meaning
0b0	The Address Comparator performs comparisons in Non-secure EL2.
0b1	The Address Comparator does not perform comparisons in Non-secure EL2.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL1, bit [13]

When Non-secure EL1 is implemented:

Non-secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Non-secure state.

EXLEVEL_NS_EL1	Meaning
0b0	The Address Comparator performs comparisons in Non-secure EL1.
0b1	The Address Comparator does not perform comparisons in Non-secure EL1.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL0, bit [12]

When Non-secure EL0 is implemented:

Non-secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Non-secure state.

EXLEVEL_NS_EL0	Meaning
0b0	The Address Comparator performs comparisons in Non-secure EL0.
0b1	The Address Comparator does not perform comparisons in Non-secure EL0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [11]

When EL3 is implemented:

EL3 address comparison control. Controls whether a comparison can occur at EL3.

EXLEVEL_S_EL3	Meaning
0b0	The Address Comparator performs comparisons at EL3.
0b1	The Address Comparator does not perform comparisons at EL3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [10]
When Secure EL2 is implemented:

Secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Secure state.

EXLEVEL_S_EL2	Meaning
0b0	The Address Comparator performs comparisons in Secure EL2.
0b1	The Address Comparator does not perform comparisons in Secure EL2.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL1, bit [9]
When Secure EL1 is implemented:

Secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Secure state.

EXLEVEL_S_EL1	Meaning
0b0	The Address Comparator performs comparisons in Secure EL1.
0b1	The Address Comparator does not perform comparisons in Secure EL1.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [8]
When Secure EL0 is implemented:

Secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Secure state.

EXLEVEL_S_EL0	Meaning
0b0	The Address Comparator performs comparisons in Secure EL0.
0b1	The Address Comparator does not perform comparisons in Secure EL0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

CONTEXT, bits [6:4]

When TRCIDR4.NUMCIDC != 0b0000 or TRCIDR4.NUMVMIDC != 0b0000:

Selects a Context Identifier Comparator or Virtual Context Identifier Comparator:

CONTEXT	Meaning	Applies when
0b000	Comparator 0.	
0b001	Comparator 1.	When UInt(TRCIDR4.NUMCIDC) > 1 or UInt(TRCIDR4.NUMVMIDC) > 1
0b010	Comparator 2.	When UInt(TRCIDR4.NUMCIDC) > 2 or UInt(TRCIDR4.NUMVMIDC) > 2
0b011	Comparator 3.	When UInt(TRCIDR4.NUMCIDC) > 3 or UInt(TRCIDR4.NUMVMIDC) > 3
0b100	Comparator 4.	When UInt(TRCIDR4.NUMCIDC) > 4 or UInt(TRCIDR4.NUMVMIDC) > 4
0b101	Comparator 5.	When UInt(TRCIDR4.NUMCIDC) > 5 or UInt(TRCIDR4.NUMVMIDC) > 5
0b110	Comparator 6.	When UInt(TRCIDR4.NUMCIDC) > 6 or UInt(TRCIDR4.NUMVMIDC) > 6
0b111	Comparator 7.	When UInt(TRCIDR4.NUMCIDC) > 7 or UInt(TRCIDR4.NUMVMIDC) > 7

The width of this field is dependent on the maximum number of Context Identifier Comparators or Virtual Context Identifier Comparators implemented. Unimplemented bits are RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CONTEXTTYPE, bits [3:2]

When TRCIDR4.NUMCIDC != 0b0000 or TRCIDR4.NUMVMIDC != 0b0000:

Controls whether the Address Comparator is dependent on a Context Identifier Comparator, a Virtual Context Identifier Comparator, or both comparisons.

CONTEXTTYPE	Meaning	Applies when
0b00	The Address Comparator is not dependent on the Context Identifier Comparators or Virtual Context Identifier Comparators.	
0b01	The Address Comparator is dependent on the Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if both the Context Identifier Comparator and the address comparison match.	When TRCIDR4.NUMCIDC != 0b0000
0b10	The Address Comparator is dependent on the Virtual Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if both the Virtual Context Identifier Comparator and the address comparison match.	When TRCIDR4.NUMVMIDC != 0b0000
0b11	The Address Comparator is dependent on the Context Identifier Comparator and Virtual Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if the Context Identifier Comparator, the Virtual Context Identifier Comparator, and address comparison all match.	When TRCIDR4.NUMCIDC != 0b0000 and TRCIDR4.NUMVMIDC != 0b0000

If TRCIDR4.NUMCIDC == 0b0000, then bit [2] is RES0.

If TRCIDR4.NUMVMIDC == 0b0000, then bit [3] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [1:0]

Reserved, RES0.

Accessing TRCACATR<n>

Must be programmed if any of the following are true:

- TRCBBCTLR.RANGE[n/2] == 1.
- TRCRSCTLR<a>.GROUP == 0b0100 and TRCRSCTLR<a>.SAC[n] == 1.
- TRCRSCTLR<a>.GROUP == 0b0101 and TRCRSCTLR<a>.ARC[n/2] == 1.
- TRCVIIECTLR.EXCLUDE[n/2] == 1.
- TRCVIIECTLR.INCLUDE[n/2] == 1.
- TRCVISSCTLR.START[n] == 1.
- TRCVISSCTLR.STOP[n] == 1.
- TRCSSCCR<>.ARC[n/2] == 1.
- TRCSSCCR<>.SAC[n] == 1.
- TRCQCTLR.RANGE[n/2] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCACATR<m> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0010	m[2:0]:0b0	0b01:m[3]

```
integer m = UInt(op2<0>:CRm<3:1>);

if m >= NUM_TRACE_ADDRESS_COMPARATOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACATR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACATR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACATR[m];
```

```
MSR TRCACATR<m>, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0010	m[2:0]:0b0	0b01:m[3]

```
integer m = UInt(op2<0>:CRm<3:1>);

if m >= NUM_TRACE_ADDRESS_COMPARATOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACATR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACATR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACATR[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCACVR<n>, Trace Address Comparator Value Register <n>, n = 0 - 15

The TRCACVR<n> characteristics are:

Purpose

Contains the address value.

Configuration

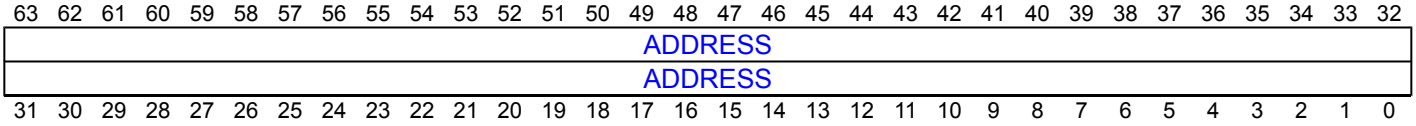
AArch64 System register TRCACVR<n> bits [63:0] are architecturally mapped to External register TRCACVR<n>[63:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMACPAIRS) * 2 > n. Otherwise, direct accesses to TRCACVR<n> are UNDEFINED.

Attributes

TRCACVR<n> is a 64-bit register.

Field descriptions



ADDRESS, bits [63:0]

Address Value.

The Address Comparators can support implementations that use multiple address widths. When the trace unit compares the ADDRESS field with an address that has a width less than this field, then the address must be zero-extended to the ADDRESS field width. The trace unit then compares all implemented bits. For example, in a system that supports both 32-bit and 64-bit addresses, when the PE is in AArch32 state the comparator must zero-extend the 32-bit address and compare against the full 64 bits that are stored in TRCACVR<n>.ADDRESS. This requires that the trace analyzer always programs all implemented bits of TRCACVR<n>.ADDRESS.

The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an UNKNOWN value, where P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The result of writing a value of all zeros or all ones to ADDRESS at bits[63:P] is the written value, and a read of the register returns the written value.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCACVR<n>

Must be programmed if any of the following are true:

- TRCBBCTLR.RANGE[n/2] == 1.
- TRCRSCTLR<a>.GROUP == 0b0100 and TRCRSCTLR<a>.SAC[n] == 1.
- TRCRSCTLR<a>.GROUP == 0b0101 and TRCRSCTLR<a>.ARC[n/2] == 1.
- TRCVIIECTLR.EXCLUDE[n/2] == 1.

- TRCVIIECTLR.INCLUDE[n/2] == 1.
- TRCVISSCTLR.START[n] == 1.
- TRCVISSCTLR.STOP[n] == 1.
- TRCSSCCR<>.ARC[n/2] == 1.
- TRCSSCCR<>.SAC[n] == 1.
- TRCQCTLR.RANGE[n/2] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCACVR<m> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0010	m[2:0]:0b0	0b00:m[3]

```
integer m = UInt(op2<0>:CRm<3:1>);

if m >= NUM_TRACE_ADDRESS_COMPARATOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACVR[m];
```

```
MSR TRCACVR<m>, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0010	m[2:0]:0b0	0b00:m[3]

```
integer m = UInt(op2<0>:CRm<3:1>);

if m >= NUM_TRACE_ADDRESS_COMPARATOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACVR[m] = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCAUTHSTATUS, Trace Authentication Status Register

The TRCAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

For additional information, see the CoreSight Architecture Specification.

Configuration

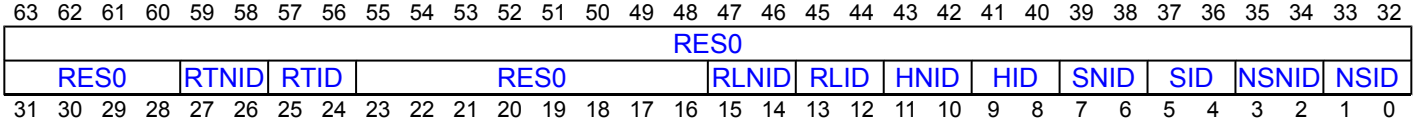
AArch64 System register TRCAUTHSTATUS bits [31:0] are architecturally mapped to External register TRCAUTHSTATUS[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCAUTHSTATUS are UNDEFINED.

Attributes

TRCAUTHSTATUS is a 64-bit register.

Field descriptions



Bits [63:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTNID.

RTID, bits [25:24]

Root invasive debug.

RTID	Meaning
0b00	Not implemented.

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLNID.

RLID, bits [13:12]

Realm invasive debug.

RLID	Meaning
0b00	Not implemented.

HNID, bits [11:10]

Hyp Non-invasive Debug. Indicates whether a separate enable control for EL2 non-invasive debug features is implemented and enabled.

HNID	Meaning
0b00	Separate Hyp non-invasive debug enable not implemented, or EL2 non-invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

HID, bits [9:8]

Hyp Invasive Debug. Indicates whether a separate enable control for EL2 invasive debug features is implemented and enabled.

HID	Meaning
0b00	Separate Hyp invasive debug enable not implemented, or EL2 invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

SNID, bits [7:6]

Secure Non-invasive Debug. Indicates whether Secure non-invasive debug features are implemented and enabled.

SNID	Meaning
0b00	Secure non-invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

When Secure state is implemented, this field reads as 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

When Secure state is not implemented, this field reads as 0b00.

SID, bits [5:4]

Secure Invasive Debug. Indicates whether Secure invasive debug features are implemented and enabled.

SID	Meaning
0b00	Secure invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

NSNID, bits [3:2]

Non-secure Non-invasive Debug. Indicates whether Non-secure non-invasive debug features are implemented and enabled.

NSNID	Meaning
0b00	Non-secure non-invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

When Non-secure state is implemented, this field reads as 0b11.

When Non-secure state is not implemented, this field reads as 0b00.

NSID, bits [1:0]

Non-secure Invasive Debug. Indicates whether Non-secure invasive debug features are implemented and enabled.

NSID	Meaning
0b00	Non-secure invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

Accessing TRCAUTHSTATUS

For implementations that support multiple access mechanisms, different access mechanisms can return different values for reads of TRCAUTHSTATUS if the authentication signals have changed and that change has not yet been synchronized by a Context synchronization event. This scenario can happen if, for example, the external debugger view is implemented separately from the system instruction view to allow for separate power domains, and so observes changes on the signals differently.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCAUTHSTATUS
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1110	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCAUTHSTATUS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUTHSTATUS;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUTHSTATUS;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUTHSTATUS;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCAUXCTLR, Trace Auxiliary Control Register

The TRCAUXCTLR characteristics are:

Purpose

The function of this register is IMPLEMENTATION DEFINED.

Configuration

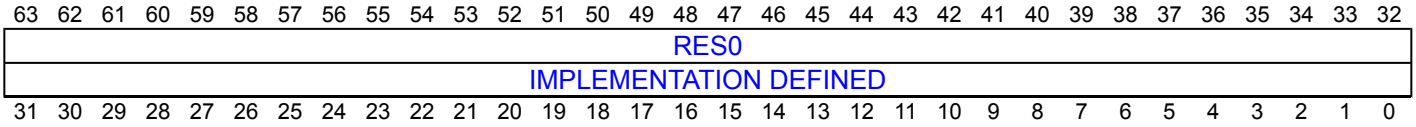
AArch64 System register TRCAUXCTLR bits [31:0] are architecturally mapped to External register TRCAUXCTLR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCAUXCTLR are UNDEFINED.

Attributes

TRCAUXCTLR is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '00000000000000000000000000000000'.

Accessing TRCAUXCTLR

If this register is nonzero then it might cause the behavior of a trace unit to contradict this architecture specification. See the documentation of the specific implementation for information about the IMPLEMENTATION DEFINED support for this register.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCAUXCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCAUXCTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUXCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUXCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUXCTLR;
```

```
MSR TRCAUXCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRCAUXCTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCAUXCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCAUXCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCAUXCTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCBBCTLR, Trace Branch Broadcast Control Register

The TRCBBCTLR characteristics are:

Purpose

Controls the regions in the memory map where branch broadcasting is active.

Configuration

AArch64 System register TRCBBCTLR bits [31:0] are architecturally mapped to External register TRCBBCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, TRCIDR0.TRCBB == 1, and UInt(TRCIDR4.NUMACPAIRS) > 0. Otherwise, direct accesses to TRCBBCTLR are UNDEFINED.

Attributes

TRCBBCTLR is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
RES0																														

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0																								MODE	RANGE[7]	RANGE[6]	RANGE[5]	RANGE[4]	RANGE[3]	RANGE[2]	RANGE[1]	RANGE[0]

Bits [63:9]

Reserved, RES0.

MODE, bit [8]

Mode.

MODE	Meaning
0b0	Exclude Mode. Branch broadcasting is not active for instructions in the address ranges defined by TRCBBCTLR.RANGE. If TRCBBCTLR.RANGE == 0x00 then branch broadcasting is active for all instructions.
0b1	Include Mode. Branch broadcasting is active for instructions in the address ranges defined by TRCBBCTLR.RANGE. If TRCBBCTLR.RANGE == 0x00 then the behavior of the trace unit is CONSTRAINED UNPREDICTABLE. That is, the trace unit might or might not consider any instructions to be in a branch broadcasting region.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE[<m>], bit [m], for m = 7 to 0

Selects whether Address Range Comparator <m> is used with branch broadcasting.

RANGE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines, is not selected.
0b1	The address range that Address Range Comparator <m> defines, is selected.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCBBCTLR

Must be programmed if TRCCONFIGR.BB == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCBBCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1111	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TRCBB == '1'
&& UInt(TRCIDR4.NUMACPAIRS) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCBBCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCBBCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCBBCTLR;
```

```
MSR TRCBBCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1111	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TRCBB == '1'
&& UInt(TRCIDR4.NUMACPAIRS) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCBBCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCBBCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCBBCTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCCCTLR, Trace Cycle Count Control Register

The TRCCCCTLR characteristics are:

Purpose

Set the threshold value for cycle counting.

Configuration

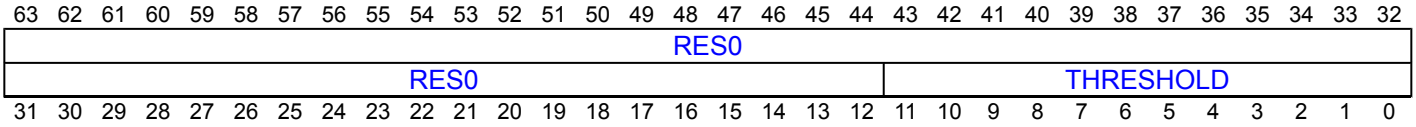
AArch64 System register TRCCCCTLR bits [31:0] are architecturally mapped to External register TRCCCCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR0.TRCCCI == 1. Otherwise, direct accesses to TRCCCCTLR are UNDEFINED.

Attributes

TRCCCCTLR is a 64-bit register.

Field descriptions



Bits [63:12]

Reserved, RES0.

THRESHOLD, bits [11:0]

Sets the threshold value for instruction trace cycle counting.

The minimum threshold value that can be programmed into THRESHOLD is given in TRCIDR3.CCITMIN. If the THRESHOLD value is smaller than the value in TRCIDR3.CCITMIN then the behavior is CONSTRAINED UNPREDICTABLE. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

Writing a value of zero when TRCCONFIGR.CCI enables instruction trace cycle counting results in CONSTRAINED UNPREDICTABLE behavior. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCCCTLR

Must be programmed if TRCCONFIGR.CCI == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCCCTLR
```

op0	op1	CRn	CRm	op2

0b10	0b001	0b0000	0b1110	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TRCCCI == '1')
then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCCCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCCCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCCCTLR;
```

MSR TRCCCCTLR, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1110	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TRCCCI == '1')
then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCCCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCCCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCCCTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDCCTLR0, Trace Context Identifier Comparator Control Register 0

The TRCCIDCCTLR0 characteristics are:

Purpose

Contains Context identifier mask values for the TRCCIDCVR<n> registers, for n = 0 to 3.

Configuration

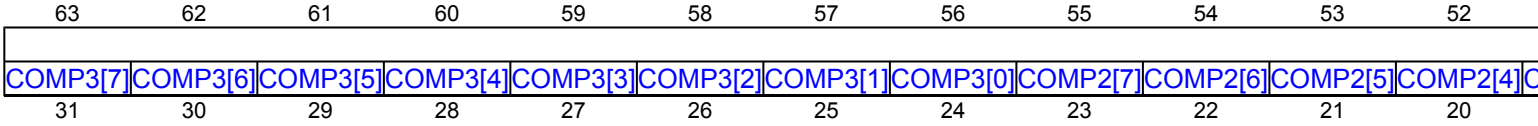
AArch64 System register TRCCIDCCTLR0 bits [31:0] are architecturally mapped to External register TRCCIDCCTLR0[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, UInt(TRCIDR4.NUMCIDC) > 0x0, and UInt(TRCIDR2.CIDSIZE) > 0. Otherwise, direct accesses to TRCCIDCCTLR0 are UNDEFINED.

Attributes

TRCCIDCCTLR0 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

COMP3[<m>], bit [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 3:

TRCCIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR3. Each bit in this field corresponds to a byte in TRCCIDCVR3.

COMP3[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR3[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR3[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP2[<m>], bit [m+16], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 2:

TRCCIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR2. Each bit in this field corresponds to a byte in TRCCIDCVR2.

COMP2[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP1[<m>], bit [m+8], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 1:

TRCCIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR1. Each bit in this field corresponds to a byte in TRCCIDCVR1.

COMP1[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP0[<m>], bit [m], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 0:

TRCCIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR0. Each bit in this field corresponds to a byte in TRCCIDCVR0.

COMP0[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Accessing TRCCIDCCTLR0

If software uses the TRCCIDCVR<n> registers, for n = 0 to 3, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCCIDCVR<n> to `0x00`.

If any bit is 1 and the relevant byte in TRCCIDCVR<n> is not `0x00`, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCIDCCTLR0
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMCIDC)
> 0x0 && UInt(TRCIDR2.CIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR0;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR0;
```

```
MSR TRCCIDCCTLR0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMCIDC)
> 0x0 && UInt(TRCIDR2.CIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDCCTLR1, Trace Context Identifier Comparator Control Register 1

The TRCCIDCCTLR1 characteristics are:

Purpose

Contains Context identifier mask values for the TRCCIDCVR<n> registers, for n = 4 to 7.

Configuration

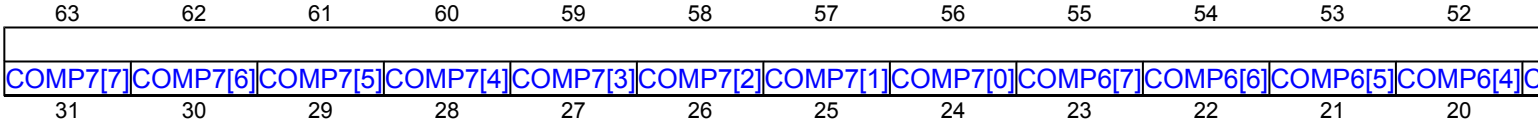
AArch64 System register TRCCIDCCTLR1 bits [31:0] are architecturally mapped to External register TRCCIDCCTLR1[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, UInt(TRCIDR4.NUMCIDC) > 0x4, and UInt(TRCIDR2.CIDSIZE) > 0. Otherwise, direct accesses to TRCCIDCCTLR1 are UNDEFINED.

Attributes

TRCCIDCCTLR1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

COMP7[<m>], bit [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 7:

TRCCIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR7. Each bit in this field corresponds to a byte in TRCCIDCVR7.

COMP7[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP6[<m>], bit [m+16], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 6:

TRCCIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR6. Each bit in this field corresponds to a byte in TRCCIDCVR6.

COMP6[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP5[<m>], bit [m+8], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 5:

TRCCIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR5. Each bit in this field corresponds to a byte in TRCCIDCVR5.

COMP5[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP4[<m>], bit [m], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 4:

TRCCIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR4. Each bit in this field corresponds to a byte in TRCCIDCVR4.

COMP4[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Accessing TRCCIDCCTLR1

If software uses the TRCCIDCVR<n> registers, for n = 4 to 7, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCCIDCVR<n> to `0x00`.

If any bit is 1 and the relevant byte in TRCCIDCVR<n> is not `0x00`, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCIDCCTLR1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMCIDC)
> 0x4 && UInt(TRCIDR2.CIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR1;
```

```
MSR TRCCIDCCTLR1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMCIDC)
> 0x4 && UInt(TRCIDR2.CIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDCVR<n>, Trace Context Identifier Comparator Value Registers <n>, n = 0 - 7

The TRCCIDCVR<n> characteristics are:

Purpose

Contains a Context identifier value.

Configuration

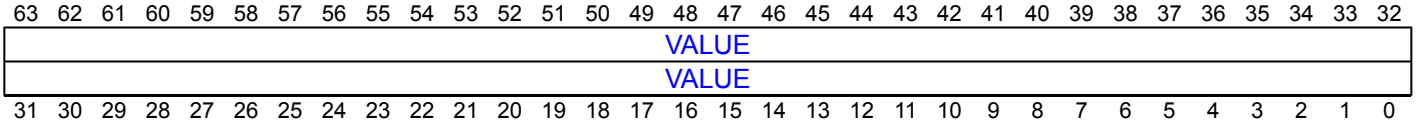
AArch64 System register TRCCIDCVR<n> bits [63:0] are architecturally mapped to External register TRCCIDCVR<n>[63:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMCIDC) > n. Otherwise, direct accesses to TRCCIDCVR<n> are UNDEFINED.

Attributes

TRCCIDCVR<n> is a 64-bit register.

Field descriptions



VALUE, bits [63:0]

Context identifier value. The width of this field is indicated by TRCIDR2.CIDSIZE. Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Context identifier is zero until the PE updates the Context identifier.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCIDCVR<n>

Must be programmed if any of the following are true:

- TRCRSCTLR<a>.GROUP == 0b0110 and TRCRSCTLR<a>.CID[n] == 1.
- TRCACATR<a>.CONTEXTTYPE == 0b01 or 0b11 and TRCACATR<a>.CONTEXT == n.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCIDCVR<m> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	m[2:0]:0b0	0b000

```
integer m = UInt(CRm<3:1>);

if m >= NUM_TRACE_CONTEXT_IDENTIFIER_COMPARATORS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCVR[m];
```

```
MSR TRCCIDCVR<m>, <Xt> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	m[2:0]:0b0	0b000

```
integer m = UInt(CRm<3:1>);

if m >= NUM_TRACE_CONTEXT_IDENTIFIER_COMPARATORS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCVR[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCLAIMCLR, Trace Claim Tag Clear Register

The TRCCLAIMCLR characteristics are:

Purpose

In conjunction with TRCCLAIMSET, provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCCLAIMCLR bits [63:0] are architecturally mapped to AArch64 System register TRCCLAIMSET[63:0].

AArch64 System register TRCCLAIMCLR bits [31:0] are architecturally mapped to External register TRCCLAIMCLR[31:0].

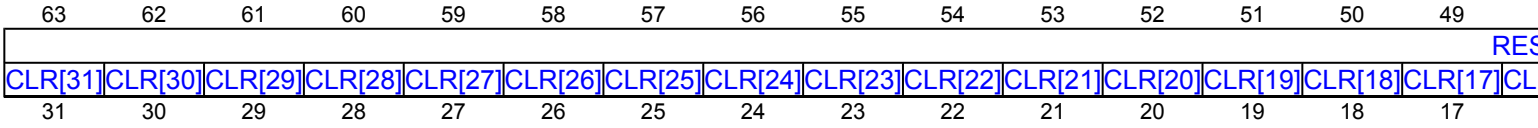
AArch64 System register TRCCLAIMCLR bits [31:0] are architecturally mapped to External register TRCCLAIMSET[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCCLAIMCLR are UNDEFINED.

Attributes

TRCCLAIMCLR is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

CLR[<m>], bit [m], for m = 31 to 0

Claim Tag Clear. Indicates the current status of Claim Tag bit <m>, and is used to clear Claim Tag bit <m> to 0.

CLR[<m>]	Meaning
0b0	On a read: Claim Tag bit <m> is not set. On a write: Ignored.
0b1	On a read: Claim Tag bit <m> is set. On a write: Clear Claim tag bit <m> to 0.

The number of Claim Tag bits implemented is indicated in TRCCLAIMSET.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0'.

Accessing this field has the following behavior:

- When m >= NUM_TRC_CLAIM_TAGS, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing TRCCLAIMCLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCLAIMCLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1001	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCLAIMCLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCLAIMCLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCLAIMCLR;
```

```
MSR TRCCLAIMCLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1001	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRCCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCLAIMCLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCLAIMCLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCLAIMCLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCLAIMSET, Trace Claim Tag Set Register

The TRCCLAIMSET characteristics are:

Purpose

In conjunction with TRCCLAIMCLR, provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCCLAIMSET bits [63:0] are architecturally mapped to AArch64 System register TRCCLAIMCLR[63:0].

AArch64 System register TRCCLAIMSET bits [31:0] are architecturally mapped to External register TRCCLAIMSET[31:0].

AArch64 System register TRCCLAIMSET bits [31:0] are architecturally mapped to External register TRCCLAIMCLR[31:0].

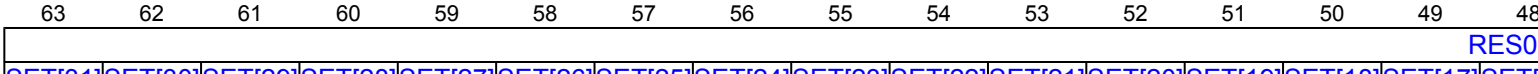
This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCCLAIMSET are UNDEFINED.

The number of claim tag bits implemented is IMPLEMENTATION DEFINED. Arm recommends that implementations support a minimum of four claim tag bits, that is, SET[3:0] reads as `0b1111`.

Attributes

TRCCLAIMSET is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

SET[<m>], bit [m], for m = 31 to 0

Claim Tag Set. Indicates whether Claim Tag bit <m> is implemented, and is used to set Claim Tag bit <m> to 1.

SET[<m>]	Meaning
`0b0`	On a read: Claim Tag bit <m> is not implemented. On a write: Ignored.
`0b1`	On a read: Claim Tag bit <m> is implemented. On a write: Set Claim Tag bit <m> to 1.

Accessing this field has the following behavior:

- When m >= NUM_TRC_CLAIM_TAGS, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RAO/W1S**.

Accessing TRCCLAIMSET

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCLAIMSET
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1000	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCLAIMSET;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCLAIMSET;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCLAIMSET;
```

```
MSR TRCCLAIMSET, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1000	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRCCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCLAIMSET = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCLAIMSET = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCLAIMSET = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCNTCTLR<n>, Trace Counter Control Register <n>, n = 0 - 3

The TRCCNTCTLR<n> characteristics are:

Purpose

Controls the operation of Counter <n>.

Configuration

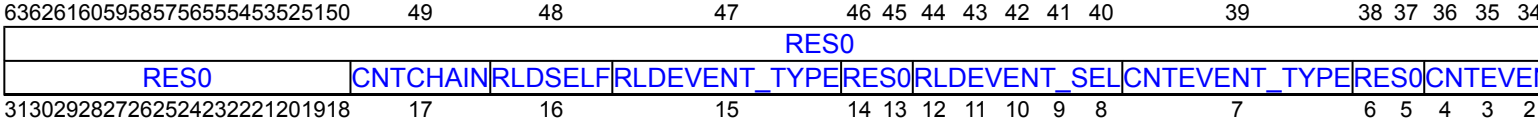
AArch64 System register TRCCNTCTLR<n> bits [31:0] are architecturally mapped to External register TRCCNTCTLR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR5.NUMCNTR) > n. Otherwise, direct accesses to TRCCNTCTLR<n> are UNDEFINED.

Attributes

TRCCNTCTLR<n> is a 64-bit register.

Field descriptions



Bits [63:18]

Reserved, RES0.

CNTCHAIN, bit [17]

When n is odd:

For TRCCNTCTLR3 and TRCCNTCTLR1, this field controls whether the Counter decrements when a reload event occurs for Counter <n-1>.

CNTCHAIN	Meaning
0b0	The Counter does not decrement when a reload event for Counter <n-1> occurs.
0b1	Counter <n> decrements when a reload event for Counter <n-1> occurs. This concatenates Counter <n> and Counter <n-1>, to provide a larger count value.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLDSELF, bit [16]

Controls whether a reload event occurs for the Counter, when the Counter reaches zero.

RLDSELF	Meaning
0b0	Normal mode. The Counter is in Normal mode.
0b1	Self-reload mode. The Counter is in Self-reload mode.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RLDEVENT_TYPE, bit [15]

Selects an event, that when it occurs causes a reload event for Counter <n>

Chooses the type of Resource Selector.

RLDEVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCCNTCTLR<n>.RLDEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCCNTCTLR<n>.RLDEVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTLR<n>.RLDEVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

RLDEVENT_SEL, bits [12:8]

Selects an event, that when it occurs causes a reload event for Counter <n>

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR<n>.RLDEVENT.TYPE controls whether TRCCNTCTLR<n>.RLDEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

CNTEVENT_TYPE, bit [7]

Selects an event, that when it occurs causes Counter <n> to decrement.

Chooses the type of Resource Selector.

CNTEVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCCNTCTLR<n>.CNTEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCCNTCTLR<n>.CNTEVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTLR<n>.CNTEVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

CNTEVENT_SEL, bits [4:0]

Selects an event, that when it occurs causes Counter <n> to decrement.

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR<n>.CNTEVENT.TYPE controls whether TRCCNTCTLR<n>.CNTEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTCTLR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCNTCTLR<m> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b01:m[1:0]	0b101

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTCTLR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTCTLR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTCTLR[m];
```

```
MSR TRCCNTCTLR<m>, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b01:m[1:0]	0b101

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTCTLR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTCTLR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTCTLR[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCNTRLDVR<n>, Trace Counter Reload Value Register <n>, n = 0 - 3

The TRCCNTRLDVR<n> characteristics are:

Purpose

This sets or returns the reload count value for Counter <n>.

Configuration

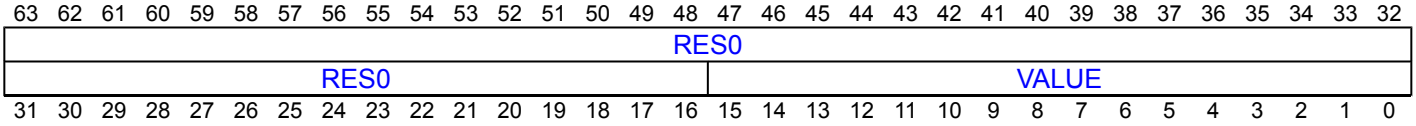
AArch64 System register TRCCNTRLDVR<n> bits [31:0] are architecturally mapped to External register TRCCNTRLDVR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR5.NUMCNTR) > n. Otherwise, direct accesses to TRCCNTRLDVR<n> are UNDEFINED.

Attributes

TRCCNTRLDVR<n> is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

VALUE, bits [15:0]

Contains the reload value for Counter <n>. When a reload event occurs for Counter <n> then the trace unit copies the VALUE<n> field into Counter <n>.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTRLDVR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCNTRLDVR<m> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b00:m[1:0]	0b101

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTRLDVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTRLDVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTRLDVR[m];
```

```
MSR TRCCNTRLDVR<m>, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b00:m[1:0]	0b101

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTRLDVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTRLDVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTRLDVR[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCNTVR<n>, Trace Counter Value Register <n>, n = 0 - 3

The TRCCNTVR<n> characteristics are:

Purpose

This sets or returns the value of Counter <n>.

Configuration

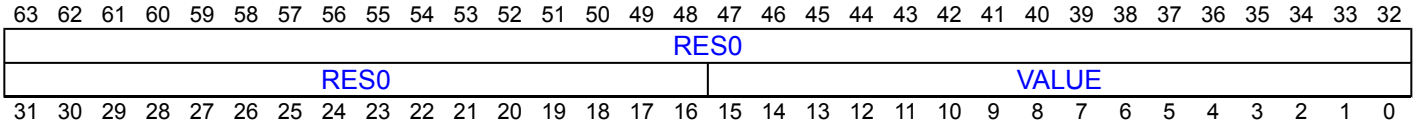
AArch64 System register TRCCNTVR<n> bits [31:0] are architecturally mapped to External register TRCCNTVR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR5.NUMCNTR) > n. Otherwise, direct accesses to TRCCNTVR<n> are UNDEFINED.

Attributes

TRCCNTVR<n> is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

VALUE, bits [15:0]

Contains the count value of Counter.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTVR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCNTVR<m> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b10:m[1:0]	0b101

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCCNTVRn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTVR[m];
```

```
MSR TRCCNTVR<m>, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b10:m[1:0]	0b101

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRCCNTVRn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTVR[m] = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCONFIGR, Trace Configuration Register

The TRCCONFIGR characteristics are:

Purpose

Controls the tracing options.

Configuration

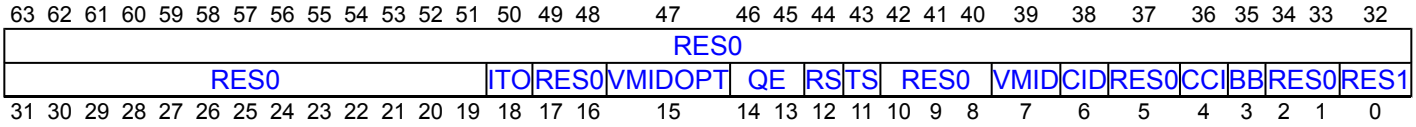
AArch64 System register TRCCONFIGR bits [31:0] are architecturally mapped to External register TRCCONFIGR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCCONFIGR are UNDEFINED.

Attributes

TRCCONFIGR is a 64-bit register.

Field descriptions



Bits [63:19]

Reserved, RES0.

ITO, bit [18]

When TRCIDR0.ITE == 1:

Instrumentation Trace Override.

ITO	Meaning
0b0	Instrumentation Trace Override disabled.
0b1	Instrumentation Trace Override enabled.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [17:16]

Reserved, RES0.

VMIDOPT, bit [15]

When TRCIDR2.VMIDOPT == 0b01:

Virtual context identifier selection control.

VMIDOPT	Meaning
0b0	VTTBR_EL2.VMID is used as the Virtual context identifier.
0b1	CONTEXTIDR_EL2.PROCID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == 0b00:

Reserved, RES0.

Virtual context identifier selection control.

VTTBR_EL2.VMID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == 0b10:

Reserved, RES1.

Virtual context identifier selection control.

CONTEXTIDR_EL2.PROCID is used as the Virtual context identifier.

Otherwise:

Reserved, RES0.

QE, bits [14:13]

When TRCIDR0.QSUPP == 0b01:

Q element generation control.

QE	Meaning
0b00	Q elements are disabled.
0b01	Q elements with instruction counts are enabled.
	Q elements without instruction counts are disabled.

All other values are reserved.

When TRCIDR0.QSUPP == 0b10:

Q element generation control.

QE	Meaning
0b00	Q elements are disabled.
0b11	Q elements with instruction counts are enabled.
	Q elements without instruction counts are enabled.

All other values are reserved.

When TRCIDR0.QSUPP == 0b11:

Q element generation control.

QE	Meaning
0b00	Q elements are disabled.
0b01	Q elements with instruction counts are enabled. Q elements without instruction counts are disabled.
0b11	Q elements with instruction counts are enabled. Q elements without instruction counts are enabled.

All other values are reserved.

Otherwise:

Reserved, RES0.

RS, bit [12]
When TRCIDR0.RETSTACK == 1:

Return stack control.

RS	Meaning
0b0	Return stack is disabled.
0b1	Return stack is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TS, bit [11]
When TRCIDR0.TSSIZE != 0b00000:

Global timestamp tracing control.

TS	Meaning
0b0	Global timestamp tracing is disabled.
0b1	Global timestamp tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [10:8]

Reserved, RES0.

VMID, bit [7]
When TRCIDR2.VMIDSIZE != 0b00000:

Virtual context identifier tracing control.

VMID	Meaning
0b0	Virtual context identifier tracing is disabled.
0b1	Virtual context identifier tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CID, bit [6]
When TRCIDR2.CIDSIZE != 0b00000:

Context identifier tracing control.

CID	Meaning
0b0	Context identifier tracing is disabled.
0b1	Context identifier tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [5]

Reserved, RES0.

CCI, bit [4]
When TRCIDR0.TRCCCI == 1:

Cycle counting instruction tracing control.

CCI	Meaning
0b0	Cycle counting instruction tracing is disabled.
0b1	Cycle counting instruction tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BB, bit [3]
When TRCIDR0.TRCBB == 1:

Branch broadcasting control.

BB	Meaning
0b0	Branch broadcasting is disabled.
0b1	Branch broadcasting is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [2:1]

Reserved, RES0.

Bit [0]

Reserved, RES1.

Accessing TRCCONFIGR

Must always be programmed.

TRCCONFIGR.QE must be set to `0b00` if TRCCONFIGR.BB is not 0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCONFIGR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCONFIGR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCONFIGR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCONFIGR;
```

```
MSR TRCCONFIGR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCONFIGR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCONFIGR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCONFIGR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVARCH, Trace Device Architecture Register

The TRCDEVARCH characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

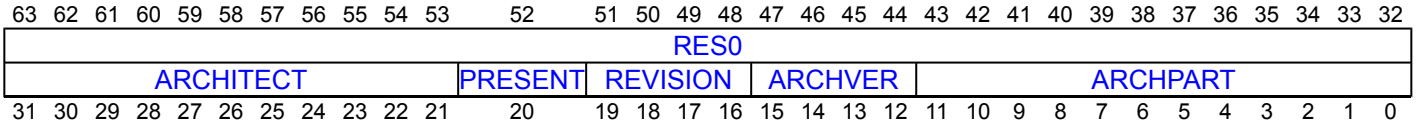
AArch64 System register TRCDEVARCH bits [31:0] are architecturally mapped to External register TRCDEVARCH[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCDEVARCH are UNDEFINED.

Attributes

TRCDEVARCH is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

ARCHITECT, bits [31:21]

Defines the architect of the component. For Trace, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, `0b0100`.

Bits [27:21] are the JEP106 identification code, `0b0111011`.

Reads as `0b01000111011`.

Access to this field is **RO**.

PRESENT, bit [20]

DEVARCH present. Indicates that the TRCDEVARCH register is present.

Reads as `0b1`.

Access to this field is **RO**.

REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

REVISION	Meaning
0b0000	ETEv1.0, FEAT_ETE.
0b0001	ETEv1.1, FEAT_ETEv1p1.
0b0010	ETEv1.2, FEAT_ETEv1p2.
0b0011	ETEv1.3, FEAT_ETEv1p3.

All other values are reserved.

Access to this field is **RO**.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

ARCHVER	Meaning
0b0101	ETEv1.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12].

This field reads as 0x5.

Access to this field is **RO**.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

ARCHPART	Meaning
0xA13	Arm PE trace architecture.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0].

This field reads as 0xA13.

Access to this field is **RO**.

Accessing TRCDEVARCH

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCDEVARCH
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1111	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVARCH;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVARCH;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVARCH;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVID, Trace Device Configuration Register

The TRCDEVID characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

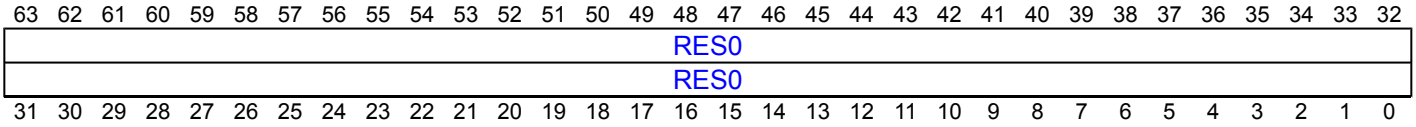
AArch64 System register TRCDEVID bits [31:0] are architecturally mapped to External register TRCDEVID[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCDEVID are UNDEFINED.

Attributes

TRCDEVID is a 64-bit register.

Field descriptions



Bits [63:0]

Reserved, RES0.

Accessing TRCDEVID

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCDEVID
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b0010	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVID;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVID;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVID;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEVENTCTL0R, Trace Event Control 0 Register

The TRCEVENTCTL0R characteristics are:

Purpose

Controls the generation of ETEEvents.

Configuration

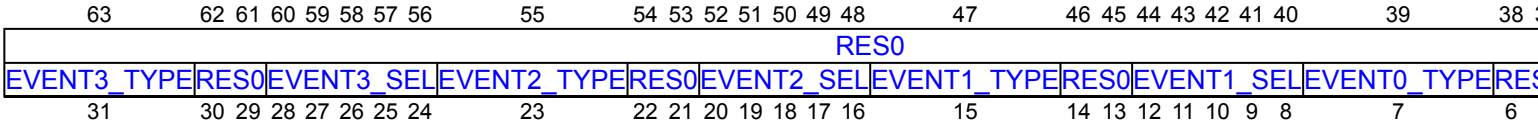
AArch64 System register TRCEVENTCTL0R bits [31:0] are architecturally mapped to External register TRCEVENTCTL0R[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR4.NUMRSPAIR != 0b0000. Otherwise, direct accesses to TRCEVENTCTL0R are UNDEFINED.

Attributes

TRCEVENTCTL0R is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

EVENT3_TYPE, bit [31]

When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 3:

Chooses the type of Resource Selector.

EVENT3_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT3.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT3.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [30:29]

Reserved, RES0.

EVENT3_SEL, bits [28:24]

When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 3:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[3] == 1, then Event element 3 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.TYPE controls whether TRCEVENTCTL0R.EVENT3.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT2_TYPE, bit [23]

When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 2:

Chooses the type of Resource Selector.

EVENT2_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT2.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT2.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [22:21]

Reserved, RES0.

EVENT2_SEL, bits [20:16]

When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 2:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[2] == 1, then Event element 2 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.TYPE controls whether TRCEVENTCTL0R.EVENT2.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT1_TYPE, bit [15]
When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 1:

Chooses the type of Resource Selector.

EVENT1_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT1.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT1.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [14:13]

Reserved, RES0.

EVENT1_SEL, bits [12:8]
When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 1:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[1] == 1, then Event element 1 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.TYPE controls whether TRCEVENTCTL0R.EVENT1.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT0_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

EVENT0_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT0.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT0.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT0_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[0] == 1, then Event element 0 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.TYPE controls whether TRCEVENTCTL0R.EVENT0.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCEVENTCTL0R

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCEVENTCTL0R
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR4.NUMRSPAIR !=
'0000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL0R;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL0R;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL0R;
```

```
MSR TRCEVENTCTL0R, <Xt>
```

op0	op1	CRn	CRm	op2

0b10	0b001	0b0000	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR4.NUMRSPAIR !=
'0000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL0R = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL0R = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL0R = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEVENTCTL1R, Trace Event Control 1 Register

The TRCEVENTCTL1R characteristics are:

Purpose

Controls the behavior of the ETEEvents that TRCEVENTCTL0R selects.

Configuration

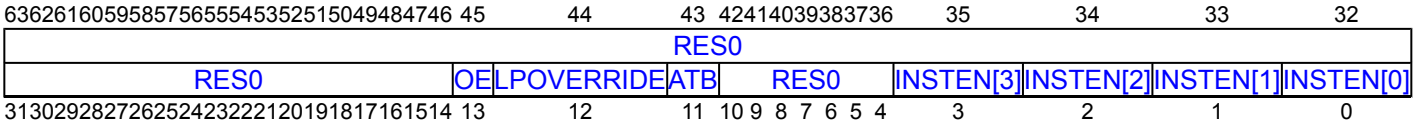
AArch64 System register TRCEVENTCTL1R bits [31:0] are architecturally mapped to External register TRCEVENTCTL1R[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCEVENTCTL1R are UNDEFINED.

Attributes

TRCEVENTCTL1R is a 64-bit register.

Field descriptions



Bits [63:14]

Reserved, RES0.

OE, bit [13]

When TRCIDR5.OE == 1:

ETE Trace Output Enable control.

OE	Meaning
0b0	Trace output to any IMPLEMENTATION DEFINED trace output interface is disabled.
0b1	Trace output to any IMPLEMENTATION DEFINED trace output interface is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

LPOVERRIDE, bit [12]

When TRCIDR5.LPOVERRIDE == 1:

Low-power Override Mode select.

LPOVERRIDE	Meaning
0b0	Trace unit Low-power Override Mode is not enabled. That is, the trace unit is permitted to enter low-power state.
0b1	Trace unit Low-power Override Mode is enabled. That is, entry to a low-power state does not affect the trace unit resources or trace generation.

Otherwise:

Reserved, RES0.

ATB, bit [11]

When TRCIDR5.ATBTRIG == 1:

AMBA Trace Bus (ATB) trigger enable.

If a CoreSight ATB interface is implemented then when ETEEvent 0 occurs the trace unit sets:

- ATID == 0x7D.
- ATDATA to the value of TRCTRACEIDR.

If the width of ATDATA is greater than the width of TRCTRACEIDR.TRACEID then the trace unit zeros the upper ATDATA bits.

If ETEEvent 0 is programmed to occur based on program execution, such as an Address Comparator, the ATB trigger might not be inserted into the ATB stream at the same time as any trace generated by that program execution is output by the trace unit. Typically, the generated trace might be buffered in a trace unit which means that the ATB trigger would be output before the associated trace is output.

If ETEEvent 0 is asserted multiple times in close succession, the trace unit is required to generate an ATB trigger for the first assertion, but might ignore one or more of the subsequent assertions. Arm recommends that the window in which ETEEvent 0 is ignored is limited only by the time taken to output an ATB trigger.

ATB	Meaning
0b0	ATB trigger is disabled.
0b1	ATB trigger is enabled.

Otherwise:

Reserved, RES0.

Bits [10:4]

Reserved, RES0.

INSTEN[<m>], bit [m], for m = 3 to 0

Event element control.

INSTEN[<m>]	Meaning
0b0	The trace unit does not generate an Event element <m>.
0b1	The trace unit generates an Event element <m> when ETEEvent <m> occurs.

Accessing this field has the following behavior:

- When TRCIDR4.NUMRSPAIR == 0b0000, access to this field is **RES0**.
- Access to this field is **RES0** if all the following are true:
  - TRCIDR4.NUMRSPAIR != 0b0000.
  - m > UInt(TRCIDR0.NUMEVENT).
- Otherwise, access to this field is **RW**.

Accessing TRCEVENTCTL1R

Must be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCEVENTCTL1R
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1001	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL1R;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL1R;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL1R;
```

```
MSR TRCEVENTCTL1R, <Xt>
```

op0	op1	CRn	CRm	op2

0b10	0b001	0b0000	0b1001	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL1R = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL1R = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL1R = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEXTINSELR<n>, Trace External Input Select Register <n>, n = 0 - 3

The TRCEXTINSELR<n> characteristics are:

Purpose

Use this to set, or read, which External Inputs are resources to the trace unit.

The name TRCEXTINSELR is an alias of TRCEXTINSELR0.

Configuration

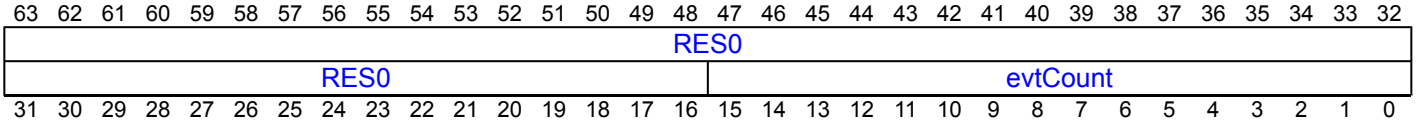
AArch64 System register TRCEXTINSELR<n> bits [31:0] are architecturally mapped to External register TRCEXTINSELR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR5.NUMEXTINSEL) > n. Otherwise, direct accesses to TRCEXTINSELR<n> are UNDEFINED.

Attributes

TRCEXTINSELR<n> is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

PMU event to select.

The event number as defined by the Arm ARM.

Software must program this field with a PMU event that is supported by the PE being programmed.

There are three ranges of PMU event numbers:

- PMU event numbers in the range `0x0000` to `0x003F` are common architectural and microarchitectural events.
- PMU event numbers in the range `0x0040` to `0x00BF` are Arm recommended common architectural and microarchitectural PMU events.
- PMU event numbers in the range `0x00C0` to `0x03FF` are IMPLEMENTATION DEFINED PMU events.

If evtCount is programmed to a PMU event that is reserved or not supported by the PE, the behavior depends on the PMU event type:

- For the range `0x0000` to `0x003F`, then the PMU event is not active, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For IMPLEMENTATION DEFINED PMU events, it is UNPREDICTABLE what PMU event, if any, is counted, and the value returned by a direct or external read of the evtCount field is UNKNOWN.

UNPREDICTABLE means the PMU event must not expose privileged information.

Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include a PMU event from a set of common IMPLEMENTATION DEFINED PMU events, then no PMU event is counted and the value read back on evtCount is the value written.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCEXTINSELR<n>

Must be programmed if any of the following is true: TRCRSCTLR<a>.GROUP == 0b0000 and TRCRSCTLR<a>.EXTIN[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCEXTINSELR<m> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b10:m[1:0]	0b100

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_EXTERNAL_INPUT_SELECTOR_RESOURCES then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEXTINSELR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEXTINSELR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEXTINSELR[m];
```

```
MSR TRCEXTINSELR<m>, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b10:m[1:0]	0b100

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_EXTERNAL_INPUT_SELECTOR_RESOURCES then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEXTINSELR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEXTINSELR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEXTINSELR[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR0, Trace ID Register 0

The TRCIDR0 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

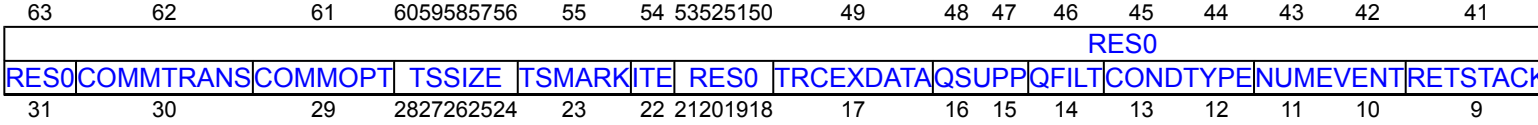
AArch64 System register TRCIDR0 bits [31:0] are architecturally mapped to External register TRCIDR0[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR0 are UNDEFINED.

Attributes

TRCIDR0 is a 64-bit register.

Field descriptions



Bits [63:31]

Reserved, RES0.

COMMTRANS, bit [30]

Transaction Start element behavior.

The value of this field is an IMPLEMENTATION DEFINED choice of:

COMMTRANS	Meaning
0b0	Transaction Start elements are P0 elements.
0b1	Transaction Start elements are not P0 elements.

Access to this field is **RO**.

COMMOPT, bit [29]

Indicates the contents and encodings of Cycle count packets.

The value of this field is an IMPLEMENTATION DEFINED choice of:

COMMOPT	Meaning
0b0	Commit mode 0.
0b1	Commit mode 1.

The Commit mode defines the contents and encodings of Cycle Count packets, in particular how Commit elements are indicated by these packets. See the descriptions of these packets for more details.

Accessing this field has the following behavior:

- Access to this field is **RAO/WI** if all the following are true:

        - TRCIDR0.TRCCCI == 1.
        - UInt(TRCIDR8.MAXSPEC) == 0x0.
    - When TRCIDR0.TRCCCI == 0, access to this field is **RAZ/WI**.
    - Otherwise, access to this field is **RO**.

TSSIZE, bits [28:24]

Indicates that the trace unit implements Global timestamping and the size of the timestamp value.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TSSIZE	Meaning
0b00000	Global timestamping not implemented.
0b01000	Global timestamping implemented with a 64-bit timestamp value.

All other values are reserved.

This field reads as 0b01000.

Access to this field is **RO**.

TSMARK, bit [23]

Indicates whether Timestamp Marker elements are generated.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TSMARK	Meaning
0b0	Timestamp Marker elements are not generated.
0b1	Timestamp Marker elements are generated.

Access to this field is **RO**.

ITE, bit [22]

Indicates whether Instrumentation Trace is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ITE	Meaning
0b0	Instrumentation Trace not implemented.
0b1	Instrumentation Trace implemented.

This field has the value 1 if FEAT_ITE is implemented.

Access to this field is **RO**.

Bits [21:18]

Reserved, RES0.

TRCEXDATA, bit [17]

When TRCIDR0.TRCDATA != 0b00:

Indicates if the trace unit implements tracing of data transfers for exceptions and exception returns. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCEXDATA	Meaning
0b0	Tracing of data transfers for exceptions and exception returns not implemented.
0b1	Tracing of data transfers for exceptions and exception returns implemented.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

QSUPP, bits [16:15]

Indicates that the trace unit implements Q element support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

QSUPP	Meaning
0b00	Q element support is not implemented.
0b01	Q element support is implemented, and only supports Q elements with instruction counts.
0b10	Q element support is implemented, and only supports Q elements without instruction counts.
0b11	Q element support is implemented, and supports: • Q elements with instruction counts. • Q elements without instruction counts.

Access to this field is **RO**.

QFILT, bit [14]

Indicates if the trace unit implements Q element filtering.

The value of this field is an IMPLEMENTATION DEFINED choice of:

QFILT	Meaning
0b0	Q element filtering is not implemented.
0b1	Q element filtering is implemented.

If TRCIDR0.QSUPP == 0b00 then this field is 0.

Access to this field is **RO**.

CONDTYPE, bits [13:12]

When TRCIDR0.TRCCOND == 1:

Indicates how conditional instructions are traced. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CONDTYPE	Meaning
0b00	Conditional instructions are traced with an indication of whether they pass or fail their condition code check.
0b01	Conditional instructions are traced with an indication of the APSR condition flags.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

NUMEVENT, bits [11:10]

When TRCIDR4.NUMRSPAIR == 0b0000:

Indicates the number of ETEEvents implemented.

NUMEVENT	Meaning
0b00	The trace unit supports 0 ETEEvents.

All other values are reserved.

Access to this field is **RO**.

When TRCIDR4.NUMRSPAIR != 0b0000:

Indicates the number of ETEEvents implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMEVENT	Meaning
0b00	The trace unit supports 1 ETEEvent.
0b01	The trace unit supports 2 ETEEvents.
0b10	The trace unit supports 3 ETEEvents.
0b11	The trace unit supports 4 ETEEvents.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

RETSTACK, bit [9]

Indicates if the trace unit supports the return stack.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RETSTACK	Meaning
0b0	Return stack not implemented.
0b1	Return stack implemented.

Access to this field is **RO**.

Bit [8]

Reserved, RES0.

TRCCCI, bit [7]

Indicates if the trace unit implements cycle counting.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCCCI	Meaning
0b0	Cycle counting not implemented.
0b1	Cycle counting implemented.

This field reads as 1.

Access to this field is **RO**.

TRCCOND, bit [6]

Indicates if the trace unit implements conditional instruction tracing. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCCOND	Meaning
0b0	Conditional instruction tracing not implemented.
0b1	Conditional instruction tracing implemented.

This field reads as 0.

Access to this field is **RO**.

TRCBB, bit [5]

Indicates if the trace unit implements branch broadcasting.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCBB	Meaning
0b0	Branch broadcasting not implemented.
0b1	Branch broadcasting implemented.

This field reads as 1.

Access to this field is **RO**.

TRCDATA, bits [4:3]

Indicates if the trace unit implements data tracing. Data tracing is not implemented in ETE and this field is reserved for other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCDATA	Meaning
0b00	Data tracing not implemented.
0b11	Data tracing implemented.

All other values are reserved.

This field reads as 0b00.

Access to this field is **RO**.

INSTP0, bits [2:1]

Indicates if load and store instructions are P0 instructions. Load and store instructions as P0 instructions is not implemented in ETE and this field is reserved for other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

INSTP0	Meaning
0b00	Load and store instructions are not P0 instructions.
0b11	Load and store instructions are P0 instructions.

All other values are reserved.

When FEAT_ETE is implemented, the only permitted value is 0b00.

Access to this field is **RO**.

Bit [0]

Reserved, RES1.

Accessing TRCIDR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR0
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1000	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR0;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR1, Trace ID Register 1

The TRCIDR1 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

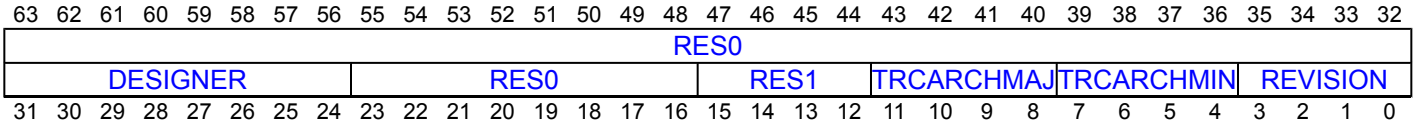
AArch64 System register TRCIDR1 bits [31:0] are architecturally mapped to External register TRCIDR1[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR1 are UNDEFINED.

Attributes

TRCIDR1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

DESIGNER, bits [31:24]

Indicates which company designed the trace unit. The permitted values of this field are the same as MIDR_EL1.Implementer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:16]

Reserved, RES0.

Bits [15:12]

Reserved, RES1.

TRCARCHMAJ, bits [11:8]

Major architecture version.

TRCARCHMAJ	Meaning
0b1111	If both TRCIDR1.TRCARCHMAJ and TRCIDR1.TRCARCHMIN == 0xF then refer to TRCDEVARCH.

All other values are reserved.

This field reads as `0b1111`.

Access to this field is **RO**.

TRCARCHMIN, bits [7:4]

Minor architecture version.

TRCARCHMIN	Meaning
`0b1111`	If both TRCIDR1.TRCARCHMAJ and TRCIDR1.TRCARCHMIN == `0xF` then refer to TRCDEVARCH.

All other values are reserved.

This field reads as `0b1111`.

Access to this field is **RO**.

REVISION, bits [3:0]

Implementation revision.

Returns an IMPLEMENTATION DEFINED value that identifies the revision of the trace unit.

Arm deprecates any use of this field and recommends that implementations set this field to zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCIDR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1001	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR1;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR10, Trace ID Register 10

The TRCIDR10 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

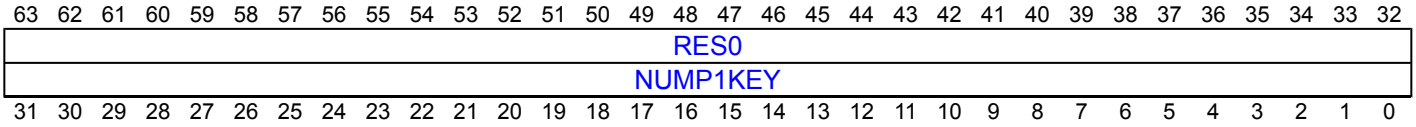
AArch64 System register TRCIDR10 bits [31:0] are architecturally mapped to External register TRCIDR10[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR10 are UNDEFINED.

Attributes

TRCIDR10 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NUMP1KEY, bits [31:0]

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR10

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR10
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0010	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR10;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR10;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR10;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR11, Trace ID Register 11

The TRCIDR11 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

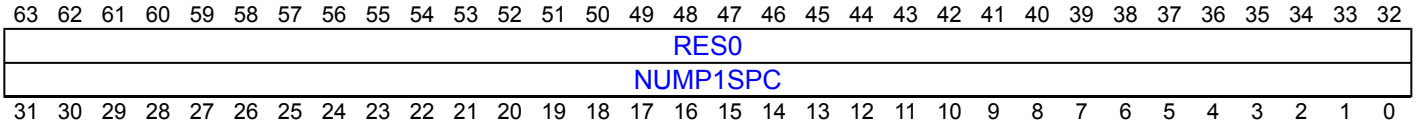
AArch64 System register TRCIDR11 bits [31:0] are architecturally mapped to External register TRCIDR11[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR11 are UNDEFINED.

Attributes

TRCIDR11 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NUMP1SPC, bits [31:0]

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of special P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR11

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR11
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0011	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR11;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR11;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR11;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR12, Trace ID Register 12

The TRCIDR12 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

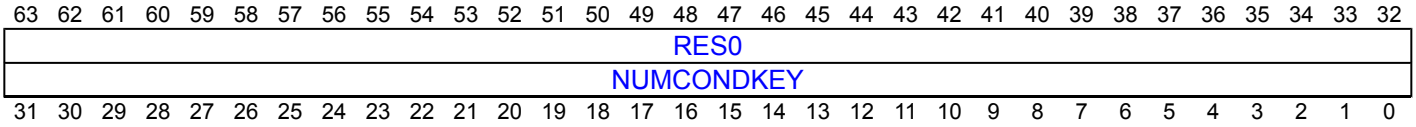
AArch64 System register TRCIDR12 bits [31:0] are architecturally mapped to External register TRCIDR12[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR12 are UNDEFINED.

Attributes

TRCIDR12 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NUMCONDKEY, bits [31:0]

When TRCIDR0.TRCCOND == 1:

Indicates the number of conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR12

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR12
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0100	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR12;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR12;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR12;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR13, Trace ID Register 13

The TRCIDR13 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

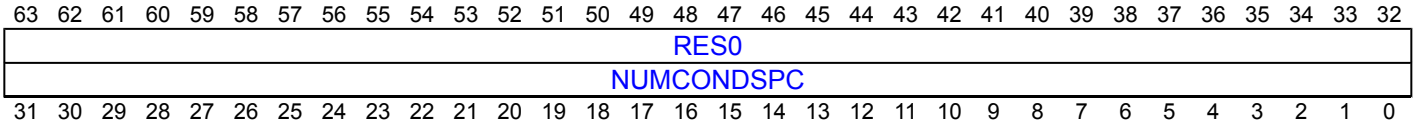
AArch64 System register TRCIDR13 bits [31:0] are architecturally mapped to External register TRCIDR13[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR13 are UNDEFINED.

Attributes

TRCIDR13 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NUMCONDSPC, bits [31:0]

When TRCIDR0.TRCCOND == 1:

Indicates the number of special conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR13

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR13
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0101	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR13;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR13;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR13;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR2, Trace ID Register 2

The TRCIDR2 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

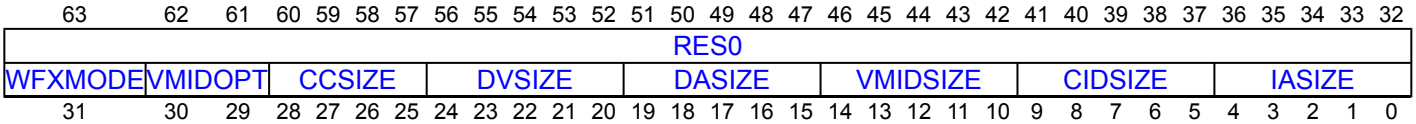
AArch64 System register TRCIDR2 bits [31:0] are architecturally mapped to External register TRCIDR2[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR2 are UNDEFINED.

Attributes

TRCIDR2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

WFXMODE, bit [31]

Indicates whether `WFI`, `WFIT`, `WFE`, and `WFET` instructions are classified as P0 instructions:

The value of this field is an IMPLEMENTATION DEFINED choice of:

WFXMODE	Meaning
`0b0`	`WFI`, `WFIT`, `WFE`, and `WFET` instructions are not classified as P0 instructions.
`0b1`	`WFI`, `WFIT`, `WFE`, and `WFET` instructions are classified as P0 instructions.

Access to this field is **RO**.

VMIDOPT, bits [30:29]

Indicates the options for Virtual context identifier selection.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMIDOPT	Meaning
`0b00`	Virtual context identifier selection not supported. TRCCONFIGR.VMIDOPT is RES0.
`0b01`	Virtual context identifier selection supported. TRCCONFIGR.VMIDOPT is implemented.
`0b10`	Virtual context identifier selection not supported. TRCCONFIGR.VMIDOPT is RES1.

All other values are reserved.

If TRCIDR2.VMIDSIZE == 0b00000 then this field is 0b00.

If TRCIDR2.VMIDSIZE != 0b00000 then this field is 0b10.

Access to this field is **RO**.

CCSIZE, bits [28:25]

When TRCIDR0.TRCCCI == 1:

Indicates the size of the cycle counter.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCSIZE	Meaning
0b0000	The cycle counter is 12 bits in length.
0b0001	The cycle counter is 13 bits in length.
0b0010	The cycle counter is 14 bits in length.
0b0011	The cycle counter is 15 bits in length.
0b0100	The cycle counter is 16 bits in length.
0b0101	The cycle counter is 17 bits in length.
0b0110	The cycle counter is 18 bits in length.
0b0111	The cycle counter is 19 bits in length.
0b1000	The cycle counter is 20 bits in length.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

DVSIZE, bits [24:20]

When TRCIDR0.TRCDATA != 0b00:

Indicates the data value size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DVSIZE	Meaning
0b00000	Data value tracing not implemented.
0b00100	Data value tracing has a maximum of 32-bit data values.
0b01000	Data value tracing has a maximum of 64-bit data values.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

DASIZE, bits [19:15]

When TRCIDR0.TRCDATA != 0b00:

Indicates the data address size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DASIZE	Meaning
0b00000	Data address tracing not implemented.
0b00100	Data address tracing has a maximum of 32-bit data addresses.
0b01000	Data address tracing has a maximum of 64-bit data addresses.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

VMIDSIZE, bits [14:10]

Indicates the trace unit Virtual context identifier size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMIDSIZE	Meaning
0b00000	Virtual context identifier tracing is not supported.
0b00001	8-bit Virtual context identifier size.
0b00010	16-bit Virtual context identifier size.
0b00100	32-bit Virtual context identifier size.

All other values are reserved.

If the PE does not implement EL2 then this field is 0b00000.

If the PE implements EL2 then this field is 0b00100.

Access to this field is **RO**.

CIDSIZE, bits [9:5]

Indicates the Context identifier size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CIDSIZE	Meaning
0b00000	Context identifier tracing is not supported.
0b00100	32-bit Context identifier size.

All other values are reserved.

This field reads as 0b00100.

Access to this field is **RO**.

IASIZE, bits [4:0]

Virtual instruction address size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IASIZE	Meaning
0b00100	Maximum of 32-bit instruction address size.
0b01000	Maximum of 64-bit instruction address size.

All other values are reserved.

This field reads as 0b01000.

Access to this field is **RO**.

Accessing TRCIDR2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR2
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1010	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR2;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR3, Trace ID Register 3

The TRCIDR3 characteristics are:

Purpose

Returns the base architecture of the trace unit.

Configuration

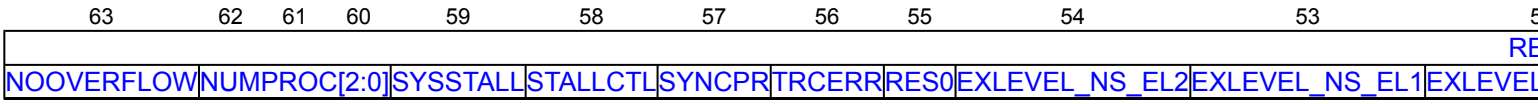
AArch64 System register TRCIDR3 bits [31:0] are architecturally mapped to External register TRCIDR3[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR3 are UNDEFINED.

Attributes

TRCIDR3 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NOOVERFLOW, bit [31]

Indicates if overflow prevention is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NOOVERFLOW	Meaning
0b0	Overflow prevention is not implemented.
0b1	Overflow prevention is implemented.

If TRCIDR3.STALLCTL == 0 then this field is 0.

Access to this field is **RO**.

NUMPROC, bits [13:12, 30:28]

Indicates the number of PEs available for tracing.

NUMPROC	Meaning
0b00000	The trace unit can trace one PE.

This field reads as 0b00000.

The NUMPROC field is split as follows:

- NUMPROC[2:0] is TRCIDR3[30:28].
- NUMPROC[4:3] is TRCIDR3[13:12].

Access to this field is **RO**.

SYSSTALL, bit [27]

Indicates if stalling of the PE is permitted.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYSSTALL	Meaning
0b0	Stalling of the PE is not permitted.
0b1	Stalling of the PE is permitted.

The value of this field might be dynamic and change based on system conditions.

If TRCIDR3.STALLCTL == 0 then this field is 0.

Access to this field is **RO**.

STALLCTL, bit [26]

Indicates if trace unit implements stalling of the PE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

STALLCTL	Meaning
0b0	Stalling of the PE is not implemented.
0b1	Stalling of the PE is implemented.

Access to this field is **RO**.

SYNCPR, bit [25]

Indicates if an implementation has a fixed synchronization period.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYNCPR	Meaning
0b0	TRCSYNCPR is read/write so software can change the synchronization period.
0b1	TRCSYNCPR is read-only so the synchronization period is fixed.

This field reads as 0.

Access to this field is **RO**.

TRCERR, bit [24]

Indicates forced tracing of System Error exceptions is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCERR	Meaning
0b0	Forced tracing of System Error exceptions is not implemented.
0b1	Forced tracing of System Error exceptions is implemented.

This field reads as 1.

Access to this field is **RO**.

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]

Indicates if Non-secure EL2 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_NS_EL2	Meaning
0b0	Non-secure EL2 is not implemented.
0b1	Non-secure EL2 is implemented.

Access to this field is **RO**.

EXLEVEL_NS_EL1, bit [21]

Indicates if Non-secure EL1 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_NS_EL1	Meaning
0b0	Non-secure EL1 is not implemented.
0b1	Non-secure EL1 is implemented.

Access to this field is **RO**.

EXLEVEL_NS_EL0, bit [20]

Indicates if Non-secure EL0 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_NS_EL0	Meaning
0b0	Non-secure EL0 is not implemented.
0b1	Non-secure EL0 is implemented.

Access to this field is **RO**.

EXLEVEL_S_EL3, bit [19]

Indicates if EL3 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL3	Meaning
0b0	EL3 is not implemented.
0b1	EL3 is implemented.

Access to this field is **RO**.

EXLEVEL_S_EL2, bit [18]

Indicates if Secure EL2 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL2	Meaning
0b0	Secure EL2 is not implemented.
0b1	Secure EL2 is implemented.

Access to this field is **RO**.

EXLEVEL_S_EL1, bit [17]

Indicates if Secure EL1 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL1	Meaning
0b0	Secure EL1 is not implemented.
0b1	Secure EL1 is implemented.

Access to this field is **RO**.

EXLEVEL_S_EL0, bit [16]

Indicates if Secure EL0 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL0	Meaning
0b0	Secure EL0 is not implemented.
0b1	Secure EL0 is implemented.

Access to this field is **RO**.

Bits [15:14]

Reserved, RES0.

CCITMIN, bits [11:0]

When TRCIDR0.TRCCCI == 0:

Indicates the minimum value that can be programmed in TRCCCCTLR.THRESHOLD.

Reads as 0x000.

Access to this field is **RO**.

When TRCIDR0.TRCCCI == 1:

Indicates the minimum value that can be programmed in TRCCCCTLR.THRESHOLD.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCITMIN	Meaning
0x001..0xFFF	The minimum value that can be programmed in TRCCCCTLR.THRESHOLD.

The minimum value of this field is 0x001.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1011	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR3;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR3;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR3;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR4, Trace ID Register 4

The TRCIDR4 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

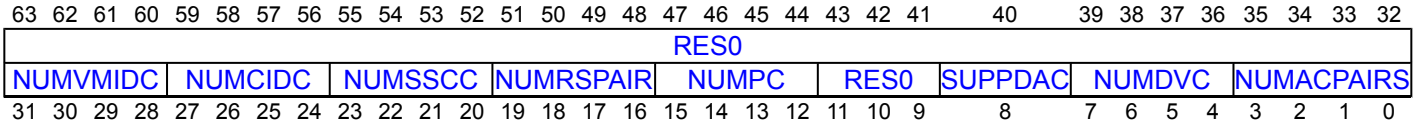
AArch64 System register TRCIDR4 bits [31:0] are architecturally mapped to External register TRCIDR4[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR4 are UNDEFINED.

Attributes

TRCIDR4 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NUMVMIDC, bits [31:28]

Indicates the number of Virtual Context Identifier Comparators that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMVMIDC	Meaning
0b0000..0b1000	The number of Virtual Context Identifier Comparators in this implementation.

All other values are reserved.

If the PE does not implement EL2 then this field is 0b0000.

Access to this field is **RO**.

NUMCIDC, bits [27:24]

Indicates the number of Context Identifier Comparators that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMCIDC	Meaning
0b0000..0b1000	The number of Context Identifier Comparators in this implementation.

All other values are reserved.

Access to this field is **RO**.

NUMSSCC, bits [23:20]

Indicates the number of Single-shot Comparator Controls that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMSSCC	Meaning
0b0000..0b1000	The number of Single-shot Comparator Controls in this implementation.

All other values are reserved.

Access to this field is **RO**.

NUMRSPAIR, bits [19:16]

Indicates the number of resource selector pairs that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMRSPAIR	Meaning
0b0000	This implementation has zero resource selector pairs.
0b0001..0b1111	The number of resource selector pairs in this implementation, minus one.

All other values are reserved.

Access to this field is **RO**.

NUMPC, bits [15:12]

Indicates the number of PE Comparator Inputs that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMPC	Meaning
0b0000..0b1000	The number of PE Comparator Inputs in this implementation.

All other values are reserved.

Access to this field is **RO**.

Bits [11:9]

Reserved, RES0.

SUPPDAC, bit [8]

When TRCIDR4.NUMACPAIRS != 0b0000:

Indicates whether data address comparisons are implemented. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SUPPDAC	Meaning
0b0	Data address comparisons not implemented.
0b1	Data address comparisons implemented.

This field reads as 0b0.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

NUMDVC, bits [7:4]

Indicates the number of data value comparators. Data value comparators are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMDVC	Meaning
0b0000..0b1000	The number of data value comparators in this implementation.

All other values are reserved.

This field reads as 0b0000.

Access to this field is **RO**.

NUMACPAIRS, bits [3:0]

Indicates the number of Address Comparator pairs that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMACPAIRS	Meaning
0b0000..0b1000	The number of Address Comparator pairs in this implementation.

All other values are reserved.

Access to this field is **RO**.

Accessing TRCIDR4

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR4
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR4;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR4;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR4;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR5, Trace ID Register 5

The TRCIDR5 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

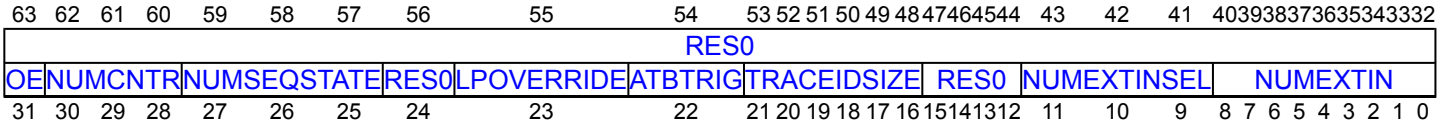
AArch64 System register TRCIDR5 bits [31:0] are architecturally mapped to External register TRCIDR5[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR5 are UNDEFINED.

Attributes

TRCIDR5 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

OE, bit [31]

Indicates support for the ETE Trace Output Enable.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OE	Meaning
0b0	ETE Trace Output Enable is not implemented.
0b1	ETE Trace Output Enable is implemented.

When FEAT_ETEv1p3 is implemented and when any IMPLEMENTATION DEFINED trace output interface is implemented, this field is 1.

Access to this field is **RO**.

NUMCNTR, bits [30:28]

Indicates the number of Counters that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMCNTR	Meaning
0b000..0b100	The number of Counters implemented.

All other values are reserved.

If TRCIDR4.NUMRSPAIR == 0b0000 then this field is 0b000.

Access to this field is **RO**.

NUMSEQSTATE, bits [27:25]

Indicates if the Sequencer is implemented and the number of Sequencer states that are implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMSEQSTATE	Meaning
0b000	The Sequencer is not implemented.
0b100	Four Sequencer states are implemented.

All other values are reserved.

If TRCIDR4.NUMRSPAIR == 0b0000 then this field is 0b000.

Access to this field is **RO**.

Bit [24]

Reserved, RES0.

LPOVERRIDE, bit [23]

Indicates support for Low-power Override Mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LPOVERRIDE	Meaning
0b0	The trace unit does not support Low-power Override Mode.
0b1	The trace unit supports Low-power Override Mode.

Access to this field is **RO**.

ATBTRIG, bit [22]

Indicates if the implementation can support ATB triggers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ATBTRIG	Meaning
0b0	The implementation does not support ATB triggers.
0b1	The implementation supports ATB triggers.

If TRCIDR4.NUMRSPAIR == 0b0000 then this field is 0.

Access to this field is **RO**.

TRACEIDSIZE, bits [21:16]

Indicates the trace ID width.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRACEIDSIZE	Meaning
0b000000	The external trace interface is not implemented.
0b000111	The implementation supports a 7-bit trace ID.

All other values are reserved.

Note

AMBA ATB requires a 7-bit trace ID width.

Access to this field is **RO**.

Bits [15:12]

Reserved, RES0.

NUMEXTINSEL, bits [11:9]

Indicates how many External Input Selector resources are implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMEXTINSEL	Meaning
0b000..0b100	The number of External Input Selector resources implemented.

All other values are reserved.

Access to this field is **RO**.

NUMEXTIN, bits [8:0]

Indicates how many External Inputs are implemented.

NUMEXTIN	Meaning
0b111111111	Unified PMU event selection.

All other values are reserved.

Access to this field is **RO**.

Accessing TRCIDR5

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR5
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1101	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR5;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR5;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR5;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR6, Trace ID Register 6

The TRCIDR6 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

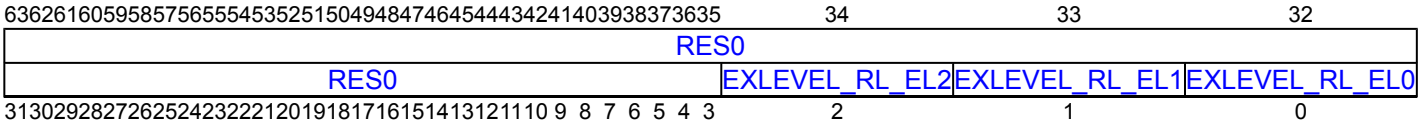
AArch64 System register TRCIDR6 bits [31:0] are architecturally mapped to External register TRCIDR6[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR6 are UNDEFINED.

Attributes

TRCIDR6 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [2]

Indicates if Realm EL2 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_RL_EL2	Meaning
0b0	Realm EL2 is not implemented.
0b1	Realm EL2 is implemented.

Access to this field is **RO**.

EXLEVEL_RL_EL1, bit [1]

Indicates if Realm EL1 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_RL_EL1	Meaning
0b0	Realm EL1 is not implemented.
0b1	Realm EL1 is implemented.

Access to this field is **RO**.

EXLEVEL_RL_EL0, bit [0]

Indicates if Realm EL0 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_RL_EL0	Meaning
0b0	Realm EL0 is not implemented.
0b1	Realm EL0 is implemented.

Access to this field is **RO**.

Accessing TRCIDR6

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR6
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR6;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR6;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR6;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR7, Trace ID Register 7

The TRCIDR7 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

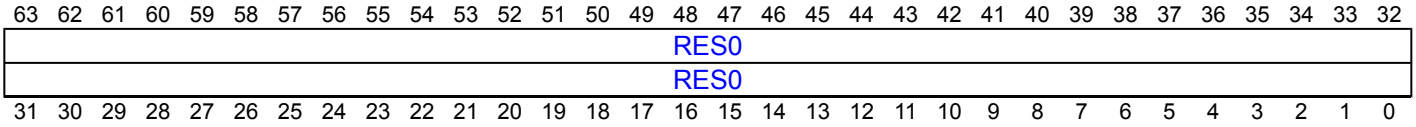
AArch64 System register TRCIDR7 bits [31:0] are architecturally mapped to External register TRCIDR7[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR7 are UNDEFINED.

Attributes

TRCIDR7 is a 64-bit register.

Field descriptions



Bits [63:0]

Reserved, RES0.

Accessing TRCIDR7

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1111	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR7;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR7;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR7;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR8, Trace ID Register 8

The TRCIDR8 characteristics are:

Purpose

Returns the maximum speculation depth of the instruction trace element stream.

Configuration

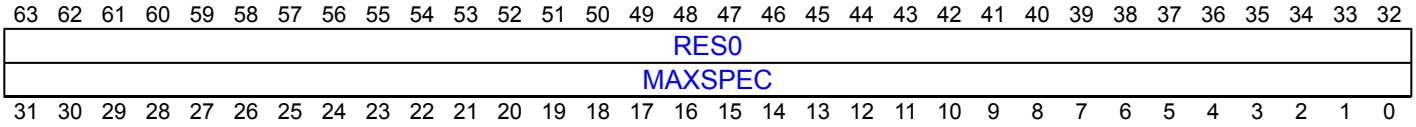
AArch64 System register TRCIDR8 bits [31:0] are architecturally mapped to External register TRCIDR8[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR8 are UNDEFINED.

Attributes

TRCIDR8 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

MAXSPEC, bits [31:0]

Indicates the maximum speculation depth of the instruction trace element stream. This is the maximum number of P0 elements in the trace element stream that can be speculative at any time.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCIDR8

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR8
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR8;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR8;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR8;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR9, Trace ID Register 9

The TRCIDR9 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

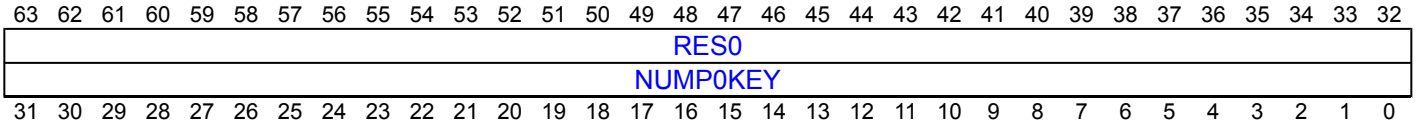
AArch64 System register TRCIDR9 bits [31:0] are architecturally mapped to External register TRCIDR9[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR9 are UNDEFINED.

Attributes

TRCIDR9 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NUMP0KEY, bits [31:0]

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P0 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR9

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR9
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR9;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR9;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR9;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIMSPEC0, Trace IMP DEF Register 0

The TRCIMSPEC0 characteristics are:

Purpose

TRCIMSPEC0 shows the presence of any IMPLEMENTATION DEFINED features, and provides an interface to enable the features that are provided.

Configuration

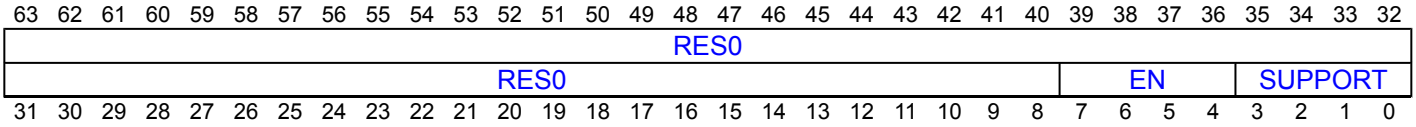
AArch64 System register TRCIMSPEC0 bits [31:0] are architecturally mapped to External register TRCIMSPEC0[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIMSPEC0 are UNDEFINED.

Attributes

TRCIMSPEC0 is a 64-bit register.

Field descriptions



Bits [63:8]

Reserved, RES0.

EN, bits [7:4]

When TRCIMSPEC0.SUPPORT != 0b0000:

Enable. Controls whether the IMPLEMENTATION DEFINED features are enabled.

EN	Meaning
0b0000	The IMPLEMENTATION DEFINED features are not enabled. The trace unit must behave as if the IMPLEMENTATION DEFINED features are not supported.
0b0001	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0010	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0011	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0100	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0101	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0110	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0111	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1000	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1001	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1010	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1011	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1100	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1101	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1110	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1111	The trace unit behavior is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0000'.

Otherwise:

Reserved, RES0.

SUPPORT, bits [3:0]

Indicates whether the implementation supports IMPLEMENTATION DEFINED features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SUPPORT	Meaning
0b0000	No IMPLEMENTATION DEFINED features are supported.
0b0001	IMPLEMENTATION DEFINED features are supported.
0b0010	IMPLEMENTATION DEFINED features are supported.
0b0011	IMPLEMENTATION DEFINED features are supported.
0b0100	IMPLEMENTATION DEFINED features are supported.
0b0101	IMPLEMENTATION DEFINED features are supported.
0b0110	IMPLEMENTATION DEFINED features are supported.
0b0111	IMPLEMENTATION DEFINED features are supported.
0b1000	IMPLEMENTATION DEFINED features are supported.
0b1001	IMPLEMENTATION DEFINED features are supported.
0b1010	IMPLEMENTATION DEFINED features are supported.
0b1011	IMPLEMENTATION DEFINED features are supported.
0b1100	IMPLEMENTATION DEFINED features are supported.
0b1101	IMPLEMENTATION DEFINED features are supported.
0b1110	IMPLEMENTATION DEFINED features are supported.
0b1111	IMPLEMENTATION DEFINED features are supported.

Use of nonzero values requires written permission from Arm.

Access to this field is **RO**.

Accessing TRCIMSPEC0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIMSPEC0
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCIMSPECn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC0;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC0;
```

```
MSR TRCIMSPEC0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRCIMSPECn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC0 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIMSPEC<n>, Trace IMP DEF Register <n>, n = 1 - 7

The TRCIMSPEC<n> characteristics are:

Purpose

These registers might return information that is specific to an implementation, or enable features specific to an implementation to be programmed. The product Technical Reference Manual describes these registers.

Configuration

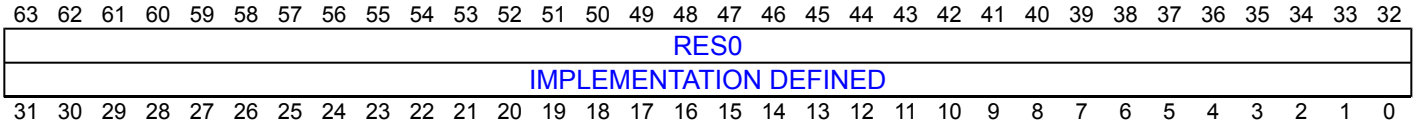
AArch64 System register TRCIMSPEC<n> bits [31:0] are architecturally mapped to External register TRCIMSPEC<n>[31:0].

This register is present only when an implementation implements TRCIMSPEC<n>, FEAT_ETE is implemented, and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIMSPEC<n> are UNDEFINED.

Attributes

TRCIMSPEC<n> is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Accessing TRCIMSPEC<n>

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIMSPEC<m> ; Where m = 1-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0:m[2:0]	0b111

```
integer m = UInt(CRm<2:0>);

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCIMSPECn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC[m];
```

```
MSR TRCIMSPEC<m>, <Xt> ; Where m = 1-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0:m[2:0]	0b111

```
integer m = UInt(CRm<2:0>);

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRCIMSPECn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC[m] = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIT, Trace Instrumentation

The TRCIT characteristics are:

Purpose

Generates an instrumentation packet in the trace.

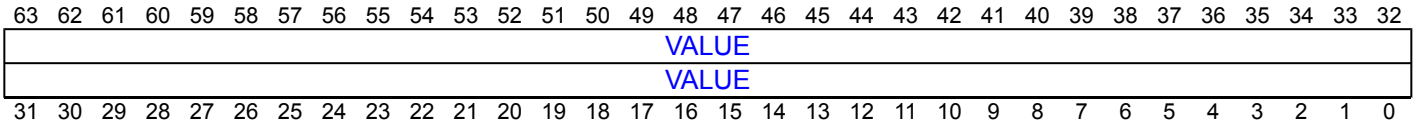
Configuration

This instruction is present only when FEAT_ITE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TRCIT are UNDEFINED.

Attributes

TRCIT is a 64-bit System instruction.

Field descriptions



VALUE, bits [63:0]

Value to be included in the Instrumentation packet.

Executing TRCIT

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TRCIT <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0010	0b111

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    AArch64.TRCIT(X[t, 64]);
elsif PSTATE.EL == EL1 then
    AArch64.TRCIT(X[t, 64]);
elsif PSTATE.EL == EL2 then
    AArch64.TRCIT(X[t, 64]);
elsif PSTATE.EL == EL3 then
    AArch64.TRCIT(X[t, 64]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCITECR_EL1, Instrumentation Trace Control Register (EL1)

The TRCITECR_EL1 characteristics are:

Purpose

Provides EL1 controls for Trace Instrumentation.

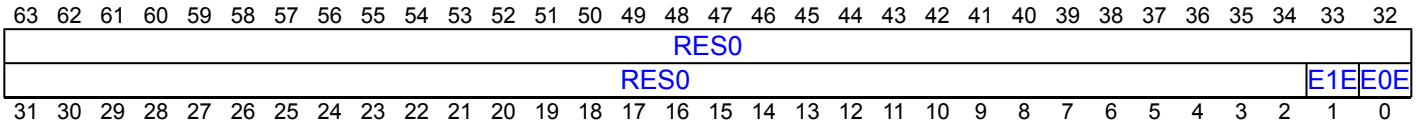
Configuration

This register is present only when FEAT_ITE is implemented, System register access to the trace unit registers is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TRCITECR_EL1 are UNDEFINED.

Attributes

TRCITECR_EL1 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

E1E, bit [1]

EL1 Instrumentation Trace Enable.

E1E	Meaning
0b0	Instrumentation trace prohibited at EL1.
0b1	Instrumentation trace not prohibited at EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0E, bit [0]

EL0 Instrumentation Trace Enable.

E0E	Meaning
0b0	Instrumentation trace prohibited at EL0.
0b1	Instrumentation trace not prohibited at EL0.

This field is ignored by the PE when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing TRCITECR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TRCITECR_EL1 or TRCITECR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCITECR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nTRCITECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x888];
    else
        X[t, 64] = TRCITECR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TRCITECR_EL2;
    else
        X[t, 64] = TRCITECR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRCITECR_EL1;
```

```
MSR TRCITECR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nTRCITECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x888] = X[t, 64];
    else
        TRCITECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TRCITECR_EL2 = X[t, 64];
    else
        TRCITECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRCITECR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, TRCITECR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x888];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TRCITECR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TRCITECR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR TRCITECR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x888] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCITECR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TRCITECR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCITECR_EL2, Instrumentation Trace Control Register (EL2)

The TRCITECR_EL2 characteristics are:

Purpose

Provides EL2 controls for Trace Instrumentation.

Configuration

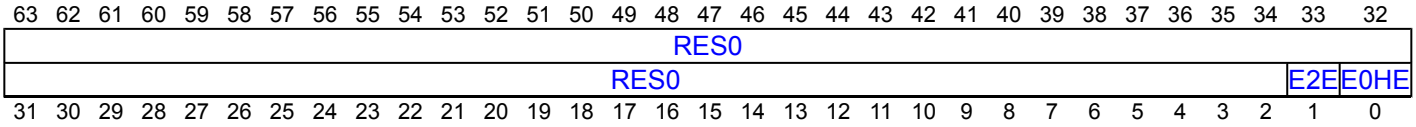
This register is present only when FEAT_ITE is implemented, System register access to the trace unit registers is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TRCITECR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

TRCITECR_EL2 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

E2E, bit [1]

EL2 Instrumentation Trace Enable.

E2E	Meaning
0b0	Instrumentation trace prohibited at EL2.
0b1	Instrumentation trace not prohibited at EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0HE, bit [0]

EL0 Instrumentation Trace Enable.

E0HE	Meaning
0b0	Instrumentation trace prohibited at EL0 when HCR_EL2.TGE == 1.
0b1	Instrumentation trace not prohibited at EL0 when HCR_EL2.TGE == 1.

This field is ignored by the PE when any of the following are true:

- HCR_EL2.TGE == 0.
- EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing TRCITECR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TRCITECR_EL2 or TRCITECR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCITECR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TRCITECR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRCITECR_EL2;
```

MSR TRCITECR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCITECR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRCITECR_EL2 = X[t, 64];
```

MRS <Xt>, TRCITECR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGRTR2_EL2.nTRCITECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x888];
    else
        X[t, 64] = TRCITECR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TRCITECR_EL2;
    else
        X[t, 64] = TRCITECR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRCITECR_EL1;
```

```
MSR TRCITECR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
'0') || HDFGWTR2_EL2.nTRCITECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x888] = X[t, 64];
    else
        TRCITECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TRCITECR_EL2 = X[t, 64];
    else
        TRCITECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRCITECR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCITEEDCR, Instrumentation Trace Extension External Debug Control Register

The TRCITEEDCR characteristics are:

Purpose

Controls instrumentation trace filtering.

Configuration

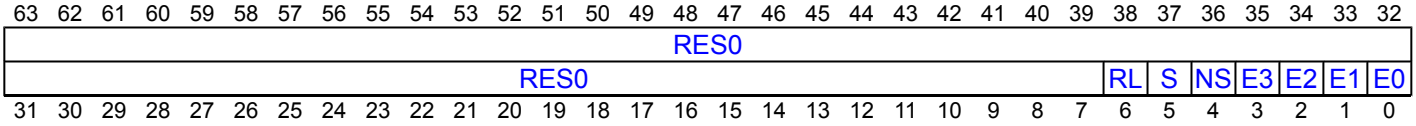
AArch64 System register TRCITEEDCR bits [31:0] are architecturally mapped to External register TRCITEEDCR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and FEAT_ITE is implemented. Otherwise, direct accesses to TRCITEEDCR are UNDEFINED.

Attributes

TRCITEEDCR is a 64-bit register.

Field descriptions



Bits [63:7]

Reserved, RES0.

RL, bit [6]

When FEAT_RME is implemented:

Instrumentation Trace in Realm state.

RL	Meaning
0b0	Instrumentation trace prohibited in Realm state.
0b1	Instrumentation trace permitted in Realm state.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

This field is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S, bit [5]

When Secure state is implemented:

Instrumentation Trace in Secure state.

S	Meaning
0b0	Instrumentation trace prohibited in Secure state.
0b1	Instrumentation trace permitted in Secure state.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

When FEAT_RME is not implemented, this field is used in conjunction with TRCCONFIGR.ITO, TRCITEEDCR.E3, and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Secure state.

When FEAT_RME is implemented, this field is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NS, bit [4]

When Any of Non-secure EL2, EL1, or EL0 are implemented:

Instrumentation Trace in Non-secure state.

NS	Meaning
0b0	Instrumentation trace prohibited in Non-secure state.
0b1	Instrumentation trace permitted in Non-secure state.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

This field is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E3, bit [3]

When EL3 is implemented:

Instrumentation Trace Enable at EL3.

E3	Meaning
0b0	Instrumentation trace prohibited at EL3.
0b1	Instrumentation trace permitted at EL3.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

When FEAT_RME is not implemented, TRCITEEDCR.E3 is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.S to control whether Instrumentation trace is permitted or prohibited at EL3.

When FEAT_RME is implemented, TRCITEEDCR.E3 is used in conjunction with TRCCONFIGR.ITO to control whether Instrumentation trace is permitted or prohibited at EL3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E<m>, bit [m], for m = 2 to 0

Instrumentation Trace Enable at EL<m>.

E<m>	Meaning
0b0	Instrumentation trace prohibited at EL<m>.
0b1	Instrumentation trace permitted at EL<m>.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

This bit is used in conjunction with TRCCONFIGR.ITO, TRCITEEDCR.NS, TRCITEEDCR.S, and TRCITEEDCR.RL to control whether Instrumentation trace is permitted or prohibited at EL<m> in the specified Security states.

TRCITEEDCR.E<2> is RES0 if EL2 is not implemented in any Security states.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCITEEDCR

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCITEEDCR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) &&
IsFeatureImplemented(FEAT_ITE)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCITEEDCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCITEEDCR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCITEEDCR;
```

MSR TRCITEEDCR, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) &&
IsFeatureImplemented(FEAT_ITE)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCITEEDCR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCITEEDCR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCITEEDCR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCOSLSR, Trace OS Lock Status Register

The TRCOSLSR characteristics are:

Purpose

Returns the status of the Trace OS Lock.

Configuration

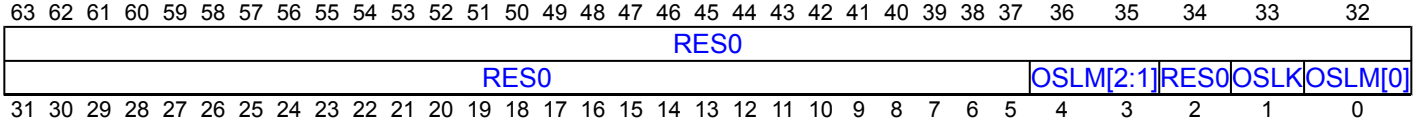
AArch64 System register TRCOSLSR bits [31:0] are architecturally mapped to External register TRCOSLSR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCOSLSR are UNDEFINED.

Attributes

TRCOSLSR is a 64-bit register.

Field descriptions



Bits [63:5]

Reserved, RES0.

OSLM, bits [4:3, 0]

OS Lock model.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OSLM	Meaning
0b000	Trace OS Lock is not implemented.
0b010	Trace OS Lock is implemented.
0b100	Trace OS Lock is not implemented, and the trace unit is controlled by the PE OS Lock.

All other values are reserved.

When FEAT_ETE is implemented, the values 0b000 and 0b010 are not permitted.

The OSLM field is split as follows:

- OSLM[2:1] is TRCOSLSR[4:3].
- OSLM[0] is TRCOSLSR[0].

Access to this field is **RO**.

Bit [2]

Reserved, RES0.

OSLK, bit [1]

OS Lock status.

OSLK	Meaning
0b0	The OS Lock is unlocked.
0b1	The OS Lock is locked.

When FEAT_ETE is implemented, this field indicates the state of the PE OS Lock.

When FEAT_ETMv4 is implemented, this field indicates the state of the Trace OS Lock.

Accessing TRCOSLSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCOSLSR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCOSLSR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCOSLSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCOSLSR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCOSLSR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPRGCTLR, Trace Programming Control Register

The TRCPRGCTLR characteristics are:

Purpose

Enables the trace unit.

Configuration

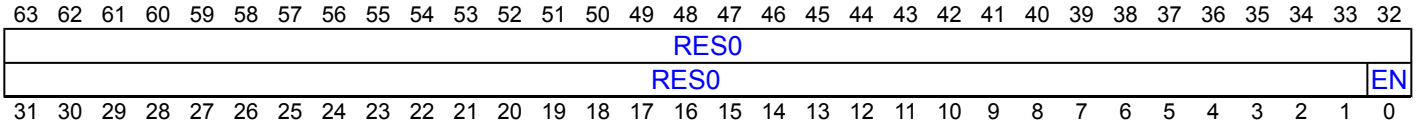
AArch64 System register TRCPRGCTLR bits [31:0] are architecturally mapped to External register TRCPRGCTLR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCPRGCTLR are UNDEFINED.

Attributes

TRCPRGCTLR is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

EN, bit [0]

Trace unit enable.

EN	Meaning
0b0	The trace unit is disabled.
0b1	The trace unit is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0'.

Accessing TRCPRGCTLR

Must be programmed.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCPRGCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCPRGCTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCPRGCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCPRGCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCPRGCTLR;
```

```
MSR TRCPRGCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRCPRGCTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCPRGCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCPRGCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCPRGCTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCQCTLR, Trace Q Element Control Register

The TRCQCTLR characteristics are:

Purpose

Controls when Q elements are enabled.

Configuration

AArch64 System register TRCQCTLR bits [31:0] are architecturally mapped to External register TRCQCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR0.QFILT == 1. Otherwise, direct accesses to TRCQCTLR are UNDEFINED.

Attributes

TRCQCTLR is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
RES0																														

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0																								MODE	RANGE[7]	RANGE[6]	RANGE[5]	RANGE[4]	RANGE[3]	RANGE[2]	RANGE[1]	RANGE[0]

Bits [63:9]

Reserved, RES0.

MODE, bit [8]

Selects whether the Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit is permitted to generate Q elements or address ranges where the trace unit is not permitted to generate Q elements:

MODE	Meaning
0b0	Exclude mode. The Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit must not generate Q elements. If no ranges are selected, Q elements are permitted across the entire memory map.
0b1	Include Mode. The Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit can generate Q elements. If all the implemented bits in RANGE are set to 0 then Q elements are disabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE[<m>], bit [m], for m = 7 to 0

Specifies whether Address Range Comparator <m> controls Q elements.

RANGE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines is not selected.
0b1	The address range that Address Range Comparator <m> defines is selected.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCQCTLR

Must be programmed if TRCCONFIGR.QE != `0b00`.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCQCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.QFILT == '1')
then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCQCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCQCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCQCTLR;
```

```
MSR TRCQCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.QFILT == '1')
then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCQCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCQCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCQCTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCRSCTLR<n>, Trace Resource Selection Control Register <n>, n = 2 - 31

The TRCRSCTLR<n> characteristics are:

Purpose

Controls the selection of the resources in the trace unit.

Configuration

AArch64 System register TRCRSCTLR<n> bits [31:0] are architecturally mapped to External register TRCRSCTLR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and (UInt(TRCIDR4.NUMRSPAIR) + 1) * 2 > n. Otherwise, direct accesses to TRCRSCTLR<n> are UNDEFINED.

Resource selector 0 always returns FALSE.

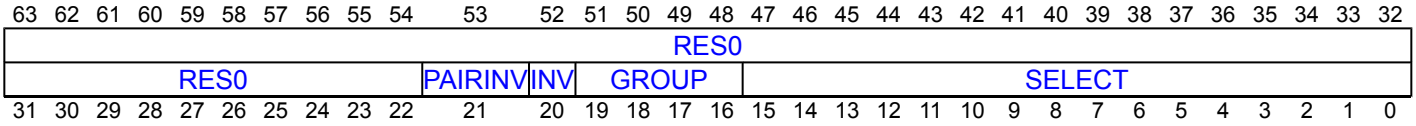
Resource selector 1 always returns TRUE.

Resource selectors are implemented in pairs. Each odd numbered resource selector is part of a pair with the even numbered resource selector that is numbered as one less than it. For example, resource selectors 2 and 3 form a pair.

Attributes

TRCRSCTLR<n> is a 64-bit register.

Field descriptions



Bits [63:22]

Reserved, RES0.

PAIRINV, bit [21]

When n is even:

Controls whether the combined result from a resource selector pair is inverted.

PAIRINV	Meaning
0b0	Do not invert the combined output of the 2 resource selectors.
0b1	Invert the combined output of the 2 resource selectors.

If:

- A is the register TRCRSCTLR<n>.
- B is the register TRCRSCTLR<n+1>.

Then the combined output of the 2 resource selectors A and B depends on the value of (A.PAIRINV, A.INV, B.INV) as follows:

- 0b000 -> A and B.
- 0b001 -> Reserved.
- 0b010 -> not(A) and B.

- 0b011 -> not(A) and not(B).
- 0b100 -> not(A) or not(B).
- 0b101 -> not(A) or B.
- 0b110 -> Reserved.
- 0b111 -> A or B.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

INV, bit [20]

Controls whether the resource, that TRCRSCTLR<n>.GROUP and TRCRSCTLR<n>.SELECT selects, is inverted.

INV	Meaning
0b0	Do not invert the output of this selector.
0b1	Invert the output of this selector.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

GROUP, bits [19:16]

Selects a group of resources.

GROUP	Meaning	SELECT
0b0000	External Input Selectors.	SELECT encoding for External Input Selectors
0b0001	PE Comparator Inputs.	SELECT encoding for PE Comparator Inputs
0b0010	Counters and Sequencer.	SELECT encoding for Counters and Sequencer
0b0011	Single-shot Comparator Controls.	SELECT encoding for Single-shot Comparator Controls
0b0100	Single Address Comparators.	SELECT encoding for Single Address Comparators
0b0101	Address Range Comparators.	SELECT encoding for Address Range Comparators
0b0110	Context Identifier Comparators.	SELECT encoding for Context Identifier Comparators
0b0111	Virtual Context Identifier Comparators.	SELECT encoding for Virtual Context Identifier Comparators

All other values are reserved.

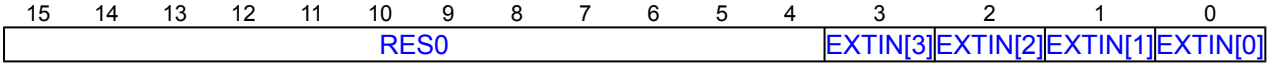
The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT, bits [15:0]

Resource Specific Controls. Contains the controls specific to the resource group selected by GROUP, described in the following sections.

SELECT encoding for External Input Selectors



Bits [15:4]

Reserved, RES0.

EXTIN[<m>], bit [m], for m = 3 to 0

Selects one or more External Inputs.

EXTIN[<m>]	Meaning
0b0	Ignore EXTIN <m>.
0b1	Select EXTIN <m>.

This bit is RES0 if m >= TRCIDR5.NUMEXTINSEL.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for PE Comparator Inputs

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0								PECOMP[7]	PECOMP[6]	PECOMP[5]	PECOMP[4]	PECOMP[3]	PECOMP[2]	PECOMP[1]	PECOMP[0]

Bits [15:8]

Reserved, RES0.

PECOMP[<m>], bit [m], for m = 7 to 0

Selects one or more PE Comparator Inputs.

PECOMP[<m>]	Meaning
0b0	Ignore PE Comparator Input <m>.
0b1	Select PE Comparator Input <m>.

This bit is RES0 if m >= TRCIDR4.NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Counters and Sequencer

15	14	13	12	11	10	9	8	7	6	5	4	3	2
RES0								SEQUENCER[3]	SEQUENCER[2]	SEQUENCER[1]	SEQUENCER[0]	COUNTERS[3]	COUNTERS[2]	COUN

Bits [15:8]

Reserved, RES0.

SEQUENCER[<m>], bit [m+4], for m = 3 to 0

Sequencer states.

SEQUENCER[<m>]	Meaning
0b0	Ignore Sequencer state <m>.
0b1	Select Sequencer state <m>.

This bit is RES0 if m >= TRCIDR5.NUMSEQSTATE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

COUNTERS[<m>], bit [m], for m = 3 to 0

Counters resources at zero.

COUNTERS[<m>]	Meaning
0b0	Ignore Counter <m>.
0b1	Select Counter <m> is zero.

This bit is RES0 if m >= TRCIDR5.NUMCNTR.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single-shot Comparator Controls

15	14	13	12	11	10	9	8	7	6	5	4	3	2
RES0								SINGLE_SHOT[7]	SINGLE_SHOT[6]	SINGLE_SHOT[5]	SINGLE_SHOT[4]	SINGLE_SHOT[3]	SINGLE_SH

Bits [15:8]

Reserved, RES0.

SINGLE_SHOT[<m>], bit [m], for m = 7 to 0

Selects one or more Single-shot Comparator Controls.

SINGLE_SHOT[<m>]	Meaning
0b0	Ignore Single-shot Comparator Control <m>.
0b1	Select Single-shot Comparator Control <m>.

This bit is RES0 if m >= TRCIDR4.NUMSSCC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single Address Comparators

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
SAC[15]	SAC[14]	SAC[13]	SAC[12]	SAC[11]	SAC[10]	SAC[9]	SAC[8]	SAC[7]	SAC[6]	SAC[5]	SAC[4]	SAC[3]	SAC[2]	SAC[1]	SA

SAC[<m>], bit [m], for m = 15 to 0

Selects one or more Single Address Comparators.

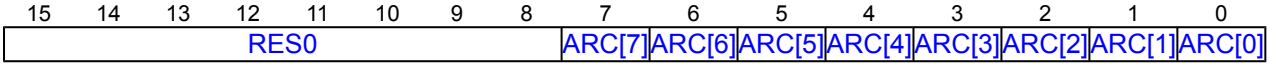
SAC[<m>]	Meaning
0b0	Ignore Single Address Comparator <m>.
0b1	Select Single Address Comparator <m>.

This bit is RES0 if m >= 2 × TRCIDR4.NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Address Range Comparators



Bits [15:8]

Reserved, RES0.

ARC[<m>], bit [m], for m = 7 to 0

Selects one or more Address Range Comparators.

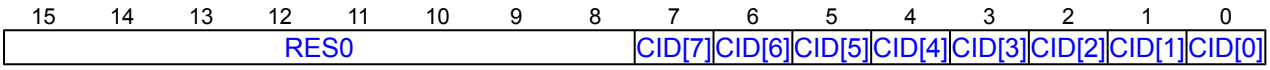
ARC[<m>]	Meaning
0b0	Ignore Address Range Comparator <m>.
0b1	Select Address Range Comparator <m>.

This bit is RES0 if m >= TRCIDR4.NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Context Identifier Comparators



Bits [15:8]

Reserved, RES0.

CID[<m>], bit [m], for m = 7 to 0

Selects one or more Context Identifier Comparators.

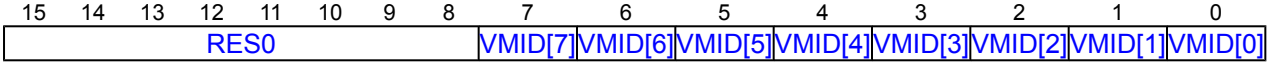
CID[<m>]	Meaning
0b0	Ignore Context Identifier Comparator <m>.
0b1	Select Context Identifier Comparator <m>.

This bit is RES0 if m >= TRCIDR4.NUMCIDC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Virtual Context Identifier Comparators



Bits [15:8]

Reserved, RES0.

VMID[<m>], bit [m], for m = 7 to 0

Selects one or more Virtual Context Identifier Comparators.

VMID[<m>]	Meaning
0b0	Ignore Virtual Context Identifier Comparator <m>.
0b1	Select Virtual Context Identifier Comparator <m>.

This bit is RES0 if m >= TRCIDR4.NUMVMIDC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCRSCTLR<n>

Must be programmed if any of the following are true:

- TRCCNTCTLR<a>.RLDEVENT.TYPE == 0 and TRCCNTCTLR<a>.RLDEVENT.SEL == n.
- TRCCNTCTLR<a>.RLDEVENT.TYPE == 1 and TRCCNTCTLR<a>.RLDEVENT.SEL == n/2.
- TRCCNTCTLR<a>.CNTEVENT.TYPE == 0 and TRCCNTCTLR<a>.CNTEVENT.SEL == n.
- TRCCNTCTLR<a>.CNTEVENT.TYPE == 1 and TRCCNTCTLR<a>.CNTEVENT.SEL == n/2.
- TRCEVENTCTL0R.EVENT0.TYPE == 0 and TRCEVENTCTL0R.EVENT0.SEL == n.
- TRCEVENTCTL0R.EVENT0.TYPE == 1 and TRCEVENTCTL0R.EVENT0.SEL == n/2.
- TRCEVENTCTL0R.EVENT1.TYPE == 0 and TRCEVENTCTL0R.EVENT1.SEL == n.
- TRCEVENTCTL0R.EVENT1.TYPE == 1 and TRCEVENTCTL0R.EVENT1.SEL == n/2.
- TRCEVENTCTL0R.EVENT2.TYPE == 0 and TRCEVENTCTL0R.EVENT2.SEL == n.
- TRCEVENTCTL0R.EVENT2.TYPE == 1 and TRCEVENTCTL0R.EVENT2.SEL == n/2.
- TRCEVENTCTL0R.EVENT3.TYPE == 0 and TRCEVENTCTL0R.EVENT3.SEL == n.
- TRCEVENTCTL0R.EVENT3.TYPE == 1 and TRCEVENTCTL0R.EVENT3.SEL == n/2.
- TRCSEQEVR<a>.B.TYPE == 0 and TRCSEQEVR<a>.B.SEL = n.
- TRCSEQEVR<a>.B.TYPE == 1 and TRCSEQEVR<a>.B.SEL = n/2.
- TRCSEQEVR<a>.F.TYPE == 0 and TRCSEQEVR<a>.F.SEL = n.
- TRCSEQEVR<a>.F.TYPE == 1 and TRCSEQEVR<a>.F.SEL = n/2.
- TRCSEQRSTEVR.RST.TYPE == 0 and TRCSEQRSTEVR.RST.SEL == n.
- TRCSEQRSTEVR.RST.TYPE == 1 and TRCSEQRSTEVR.RST.SEL == n/2.
- TRCTSCTLR.EVENT.TYPE == 0 and TRCTSCTLR.EVENT.SEL == n.
- TRCTSCTLR.EVENT.TYPE == 1 and TRCTSCTLR.EVENT.SEL == n/2.
- TRCVICTLR.EVENT.TYPE == 0 and TRCVICTLR.EVENT.SEL == n.
- TRCVICTLR.EVENT.TYPE == 1 and TRCVICTLR.EVENT.SEL == n/2.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCRSCTLR<m> ; Where m = 2-31
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	m[3:0]	0b00:m[4]

```
integer m = UInt(op2<0>:CRm<3:0>);

if m >= NUM_TRACE_RESOURCE_SELECTOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSCTLR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSCTLR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSCTLR[m];
```

```
MSR TRCRSCTLR<m>, <Xt> ; Where m = 2-31
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	m[3:0]	0b00:m[4]

```
integer m = UInt(op2<0>:CRm<3:0>);

if m >= NUM_TRACE_RESOURCE_SELECTOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSCTLR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSCTLR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSCTLR[m] = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCRSR, Trace Resources Status Register

The TRCRSR characteristics are:

Purpose

Use this to set, or read, the status of the resources.

Configuration

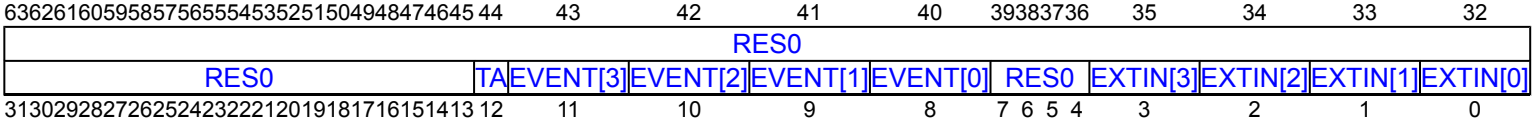
AArch64 System register TRCRSR bits [31:0] are architecturally mapped to External register TRCRSR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCRSR are UNDEFINED.

Attributes

TRCRSR is a 64-bit register.

Field descriptions



Bits [63:13]

Reserved, RES0.

TA, bit [12]

Tracing active.

TA	Meaning
0b0	Tracing is not active.
0b1	Tracing is active.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

EVENT[<m>], bit [m+8], for m = 3 to 0

Untraced status of ETEEvents.

EVENT[<m>]	Meaning
0b0	An ETEEvent <m> has not occurred.
0b1	An ETEEvent <m> has occurred while the resources were in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When TRCIDR4.NUMRSPAIR == 0b0000, access to this field is **RES0**.

- Access to this field is **RES0** if all the following are true:
    - TRCIDR4.NUMRSPAIR != 0b0000.
    - m > UInt(TRCIDR0.NUMEVENT).
- Otherwise, access to this field is **RW**.

Bits [7:4]

Reserved, RES0.

EXTIN[<m>], bit [m], for m = 3 to 0

The sticky status of the External Input Selectors.

EXTIN[<m>]	Meaning
0b0	An event selected by External Input Selector <m> has not occurred.
0b1	At least one event selected by External Input Selector <m> has occurred while the resources were in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR5.NUMEXTINSEL), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCRSR

Must always be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCRSR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1010	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSR;
```

```
MSR TRCRSR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1010	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQEVR<n>, Trace Sequencer State Transition Control Register <n>, n = 0 - 2

The TRCSEQEVR<n> characteristics are:

Purpose

Moves the Sequencer state:

- Backwards, from state n+1 to state n when a programmed resource event occurs.
- Forwards, from state n to state n+1 when a programmed resource event occurs.

Configuration

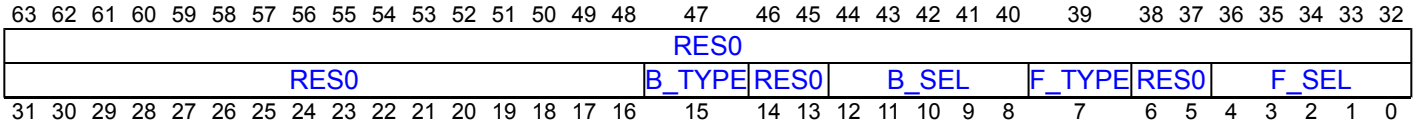
AArch64 System register TRCSEQEVR<n> bits [31:0] are architecturally mapped to External register TRCSEQEVR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQEVR<n> are UNDEFINED.

Attributes

TRCSEQEVR<n> is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

B_TYPE, bit [15]

Backward field. Selects an event that causes the Sequencer to move from state n+1 to state n. For example, if TRCSEQEVR2.B.SEL == `0x14` then when event `0x14` occurs, the Sequencer moves from state 3 to state 2.

Chooses the type of Resource Selector.

B_TYPE	Meaning
`0b0`	A single Resource Selector. TRCSEQEVR<n>.B.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
`0b1`	A Boolean-combined pair of Resource Selectors. TRCSEQEVR<n>.B.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR<n>.B.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

B_SEL, bits [12:8]

Backward field. Selects an event that causes the Sequencer to move from state n+1 to state n. For example, if TRCSEQEVR2.B.SEL == `0x14` then when event `0x14` occurs, the Sequencer moves from state 3 to state 2.

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.B.TYPE controls whether TRCSEQEVR<n>.B.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

F_TYPE, bit [7]

Forward field. Selects an event that causes the Sequencer to move from state n to state n+1. For example, if TRCSEQEVR1.F.SEL == `0x12` then when event `0x12` occurs, the Sequencer moves from state 1 to state 2.

Chooses the type of Resource Selector.

F_TYPE	Meaning
`0b0`	A single Resource Selector. TRCSEQEVR<n>.F.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
`0b1`	A Boolean-combined pair of Resource Selectors. TRCSEQEVR<n>.F.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR<n>.F.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

F_SEL, bits [4:0]

Forward field. Selects an event that causes the Sequencer to move from state n to state n+1. For example, if TRCSEQEVR1.F.SEL == `0x12` then when event `0x12` occurs, the Sequencer moves from state 1 to state 2.

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.F.TYPE controls whether TRCSEQEVR<n>.F.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQEVR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.SEQUENCER != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSEQEVR<m> ; Where m = 0-2
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b00:m[1:0]	0b100

```
integer m = UInt(CRm<1:0>);

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQEVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQEVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQEVR[m];
```

```
MSR TRCSEQEVR<m>, <Xt> ; Where m = 0-2
```

op0	op1	CRn	CRm	op2

0b10	0b001	0b0000	0b00:m[1:0]	0b100

```
integer m = UInt(CRm<1:0>);

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQEVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQEVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQEVR[m] = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQRSTEVR, Trace Sequencer Reset Control Register

The TRCSEQRSTEVR characteristics are:

Purpose

Moves the Sequencer to state 0 when a programmed resource event occurs.

Configuration

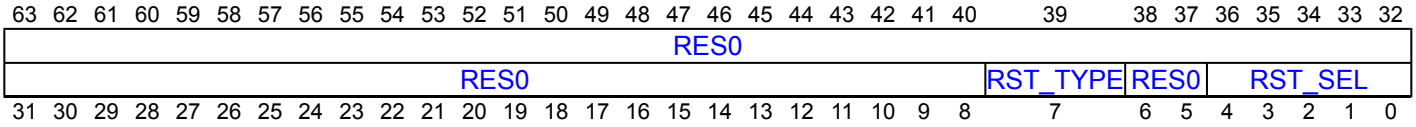
AArch64 System register TRCSEQRSTEVR bits [31:0] are architecturally mapped to External register TRCSEQRSTEVR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQRSTEVR are UNDEFINED.

Attributes

TRCSEQRSTEVR is a 64-bit register.

Field descriptions



Bits [63:8]

Reserved, RES0.

RST_TYPE, bit [7]

Reset field. Selects an event that causes the Sequencer to move to state 0.

Chooses the type of Resource Selector.

RST_TYPE	Meaning
0b0	A single Resource Selector. TRCSEQRSTEVR.RST.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCSEQRSTEVR.RST.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQRSTEVR.RST.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

RST_SEL, bits [4:0]

Reset field. Selects an event that causes the Sequencer to move to state 0.

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQRSTEVR.RST.TYPE controls whether TRCSEQRSTEVR.RST.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQRSTEVR

Must be programmed if TRCRSCTLR<a>.GROUP == `0b0010` and TRCRSCTLR<a>.SEQUENCER != `0b0000`.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSEQRSTEVR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0110	0b100

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR5.NUMSEQSTATE !=
'000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQRSTEVR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQRSTEVR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQRSTEVR;
```

```
MSR TRCSEQRSTEVR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0110	0b100

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR5.NUMSEQSTATE !=
'000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQRSTEVR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQRSTEVR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQRSTEVR = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQSTR, Trace Sequencer State Register

The TRCSEQSTR characteristics are:

Purpose

Use this to set, or read, the Sequencer state.

Configuration

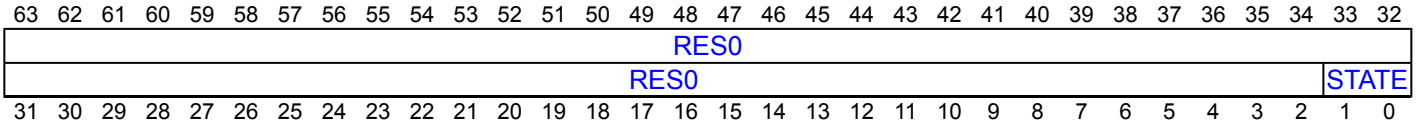
AArch64 System register TRCSEQSTR bits [31:0] are architecturally mapped to External register TRCSEQSTR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQSTR are UNDEFINED.

Attributes

TRCSEQSTR is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

STATE, bits [1:0]

Set or returns the state of the Sequencer.

STATE	Meaning
0b00	State 0.
0b01	State 1.
0b10	State 2.
0b11	State 3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQSTR

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.SEQUENCER != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCSEQSTR

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0111	0b100

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR5.NUMSEQSTATE !=
'000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCSEQSTR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQSTR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQSTR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQSTR;
```

MSR TRCSEQSTR, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0111	0b100

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR5.NUMSEQSTATE !=
'000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRCSEQSTR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQSTR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQSTR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQSTR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSCCR<n>, Trace Single-shot Comparator Control Register <n>, n = 0 - 7

The TRCSSCCR<n> characteristics are:

Purpose

Controls the corresponding Single-shot Comparator Control resource.

Configuration

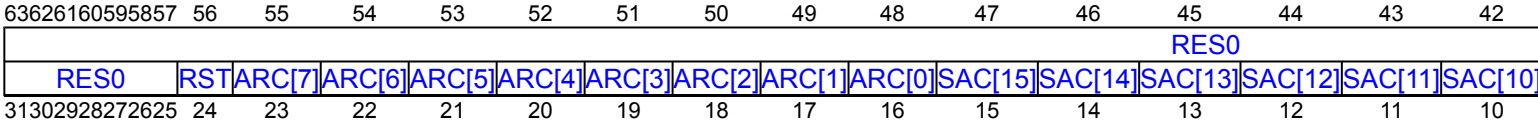
AArch64 System register TRCSSCCR<n> bits [31:0] are architecturally mapped to External register TRCSSCCR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMSSCC) > n. Otherwise, direct accesses to TRCSSCCR<n> are UNDEFINED.

Attributes

TRCSSCCR<n> is a 64-bit register.

Field descriptions



Bits [63:25]

Reserved, RES0.

RST, bit [24]

Selects the Single-shot Comparator Control mode.

RST	Meaning
0b0	The Single-shot Comparator Control is in single-shot mode.
0b1	The Single-shot Comparator Control is in multi-shot mode.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

ARC[<m>], bit [m+16], for m = 7 to 0

Selects one or more Address Range Comparators for Single-shot control.

ARC[<m>]	Meaning
0b0	The Address Range Comparator <m>, is not selected for Single-shot control.
0b1	The Address Range Comparator <m>, is selected for Single-shot control.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

SAC[<m>], bit [m], for m = 15 to 0

Selects one or more Single Address Comparators for Single-shot control.

SAC[<m>]	Meaning
0b0	The Single Address Comparator <m>, is not selected for Single-shot control.
0b1	The Single Address Comparator <m>, is selected for Single-shot control.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS) * 2, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCSSCCR<n>

Must be programmed if any TRCRSCTLR<a>.GROUP == 0b0011 and TRCRSCTLR<a>.SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSSCCR<m> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b0:m[2:0]	0b010

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCCR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCCR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCCR[m];
```

```
MSR TRCSSCCR<m>, <Xt> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b0:m[2:0]	0b010

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCCR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCCR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCCR[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSCSR<n>, Trace Single-shot Comparator Control Status Register <n>, n = 0 - 7

The TRCSSCSR<n> characteristics are:

Purpose

Returns the status of the corresponding Single-shot Comparator Control.

Configuration

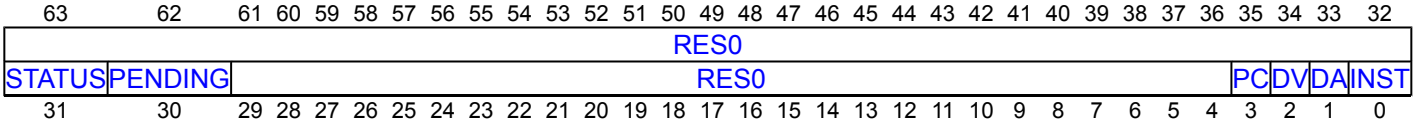
AArch64 System register TRCSSCSR<n> bits [31:0] are architecturally mapped to External register TRCSSCSR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMSSCC) > n. Otherwise, direct accesses to TRCSSCSR<n> are UNDEFINED.

Attributes

TRCSSCSR<n> is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

STATUS, bit [31]

Single-shot Comparator Control status. Indicates if any of the comparators selected by this Single-shot Comparator control have matched. The selected comparators are defined by TRCSSCCR<n>.ARC, TRCSSCCR<n>.SAC, and TRCSSPCICR<n>.PC.

STATUS	Meaning
0b0	No match has occurred. When the first match occurs, this field takes a value of 1. It remains at 1 until explicitly modified by a write to this register.
0b1	One or more matches has occurred. If TRCSSCCR<n>.RST == 0 then: • There is only one match and no more matches are possible. • Software must reset this field to 0 to re-enable the Single-shot Comparator Control.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

PENDING, bit [30]

Single-shot pending status. The Single-shot Comparator Control fired while the resources were in the Paused state.

PENDING	Meaning
0b0	No match has occurred.
0b1	One or more matches has occurred.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [29:4]

Reserved, RES0.

PC, bit [3]

PE Comparator Input support. Indicates if the Single-shot Comparator Control supports PE Comparator Inputs.

PC	Meaning
0b0	This Single-shot Comparator Control does not support PE Comparator Inputs. Selecting any PE Comparator Inputs using the associated TRCSSPCICR<n> results in CONSTRAINED UNPREDICTABLE behavior of the Single-shot Comparator Control resource. The Single-shot Comparator Control might match unexpectedly or might not match.
0b1	This Single-shot Comparator Control supports PE Comparator Inputs.

Access to this field is **RO**.

DV, bit [2]

Data value comparator support. Data value comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

DV	Meaning
0b0	This Single-shot Comparator Control does not support data value comparisons.
0b1	This Single-shot Comparator Control supports data value comparisons.

This field reads as 0.

Access to this field is **RO**.

DA, bit [1]

Data Address Comparator support. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

DA	Meaning
0b0	This Single-shot Comparator Control does not support data address comparisons.
0b1	This Single-shot Comparator Control supports data address comparisons.

This field reads as 0.

Access to this field is **RO**.

INST, bit [0]

Instruction Address Comparator support. Indicates if the Single-shot Comparator Control supports instruction address comparisons.

INST	Meaning
0b0	This Single-shot Comparator Control does not support instruction address comparisons.
0b1	This Single-shot Comparator Control supports instruction address comparisons.

This field reads as 1.

Access to this field is **RO**.

Accessing TRCSSCSR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0011 and TRCRSCTLR<a>.SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSSCSR<m> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b1:m[2:0]	0b010

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCSSCSRn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCSR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCSR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCSR[m];
```

```
MSR TRCSSCSR<m>, <Xt> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b1:m[2:0]	0b010

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRCSSCSRn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCSR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCSR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCSR[m] = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSPCICR<n>, Trace Single-shot Processing Element Comparator Input Control Register <n>, n = 0 - 7

The TRCSSPCICR<n> characteristics are:

Purpose

Returns the status of the corresponding Single-shot Comparator Control.

Configuration

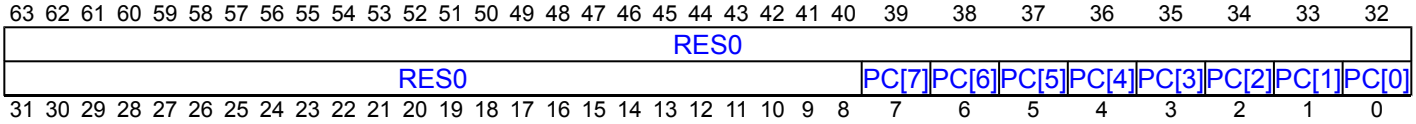
AArch64 System register TRCSSPCICR<n> bits [31:0] are architecturally mapped to External register TRCSSPCICR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, UInt(TRCIDR4.NUMSSCC) > n, UInt(TRCIDR4.NUMPC) > 0, and TRCSSCSR<n>.PC == 1. Otherwise, direct accesses to TRCSSPCICR<n> are UNDEFINED.

Attributes

TRCSSPCICR<n> is a 64-bit register.

Field descriptions



Bits [63:8]

Reserved, RES0.

PC[<m>], bit [m], for m = 7 to 0

Selects one or more PE Comparator Inputs for Single-shot control.

PC[<m>]	Meaning
0b0	The single PE Comparator Input <m>, is not selected as for Single-shot control.
0b1	The single PE Comparator Input <m>, is selected as for Single-shot control.

This bit is RES0 if m >= TRCIDR4.NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSSPCICR<n>

Must be programmed if implemented and any TRCRSCTLR<a>.GROUP == 0b0011 and TRCRSCTLR<a>.SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSSPCICR<m> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b0:m[2:0]	0b011

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSPCICR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSPCICR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSPCICR[m];
```

```
MSR TRCSSPCICR<m>, <Xt> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b0:m[2:0]	0b011

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSPCICR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSPCICR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSPCICR[m] = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSTALLCTLR, Trace Stall Control Register

The TRCSTALLCTLR characteristics are:

Purpose

Enables trace unit functionality that prevents trace unit buffer overflows.

Configuration

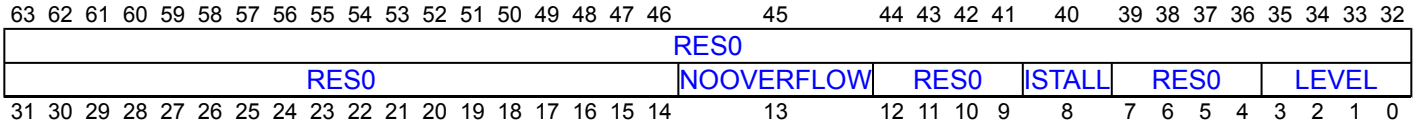
AArch64 System register TRCSTALLCTLR bits [31:0] are architecturally mapped to External register TRCSTALLCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR3.STALLCTL == 1. Otherwise, direct accesses to TRCSTALLCTLR are UNDEFINED.

Attributes

TRCSTALLCTLR is a 64-bit register.

Field descriptions



Bits [63:14]

Reserved, RES0.

NOOVERFLOW, bit [13]

When TRCIDR3.NOOVERFLOW == 1:

Trace overflow prevention.

NOOVERFLOW	Meaning
0b0	Trace unit buffer overflow prevention is disabled.
0b1	Trace unit buffer overflow prevention is enabled.

Note

Enabling this feature might cause a significant performance impact.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:9]

Reserved, RES0.

ISTALL, bit [8]

Instruction stall control. Controls if a trace unit can stall the PE when the trace buffer space is less than LEVEL.

ISTALL	Meaning
0b0	The trace unit must not stall the PE.
0b1	The trace unit can stall the PE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

LEVEL, bits [3:0]

Threshold level field. The field can support 16 monotonic levels from 0b0000 to 0b1111.

The value 0b0000 defines the Minimal invasion level. This setting has a greater risk of a trace unit buffer overflow.

The value 0b1111 defines the Maximum invasion level. This setting has a reduced risk of a trace unit buffer overflow.

Note

For some implementations, invasion might occur at the minimal invasion level.

One or more of the least significant bits of LEVEL are permitted to be RES0. Arm recommends that LEVEL[3:2] are fully implemented. Arm strongly recommends that LEVEL[3] is always implemented. If one or more bits are RES0 and are written with a nonzero value, the effective value of LEVEL is rounded down to the nearest power of 2 value which has the RES0 bits as zero. For example, if LEVEL[1:0] are RES0 and a value of 0b1110 is written to LEVEL, the effective value of LEVEL is 0b1100.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSTALLCTLR

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSTALLCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1011	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR3.STALLCTL ==
'1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTALLCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTALLCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTALLCTLR;
```

```
MSR TRCSTALLCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1011	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR3.STALLCTL ==
'1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSTALLCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSTALLCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSTALLCTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSTATR, Trace Status Register

The TRCSTATR characteristics are:

Purpose

Returns the trace unit status.

Configuration

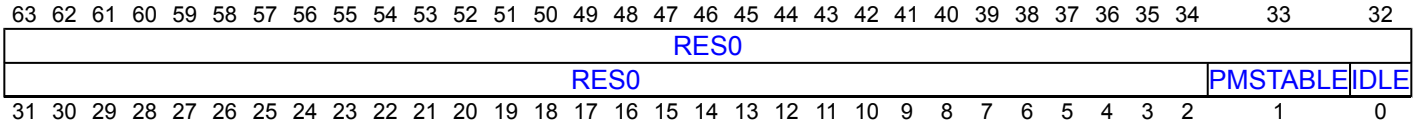
AArch64 System register TRCSTATR bits [31:0] are architecturally mapped to External register TRCSTATR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCSTATR are UNDEFINED.

Attributes

TRCSTATR is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

PMSTABLE, bit [1]

Programmers' model stable.

PMSTABLE	Meaning
0b0	The programmers' model is not stable.
0b1	The programmers' model is stable.

Accessing this field has the following behavior:

- When the trace unit is enabled, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

IDLE, bit [0]

Idle status.

IDLE	Meaning
0b0	The trace unit is not idle.
0b1	The trace unit is idle.

Accessing TRCSTATR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSTATR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCSTATR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTATR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTATR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTATR;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSYNCPR, Trace Synchronization Period Register

The TRCSYNCPR characteristics are:

Purpose

Controls how often trace protocol synchronization requests occur.

Configuration

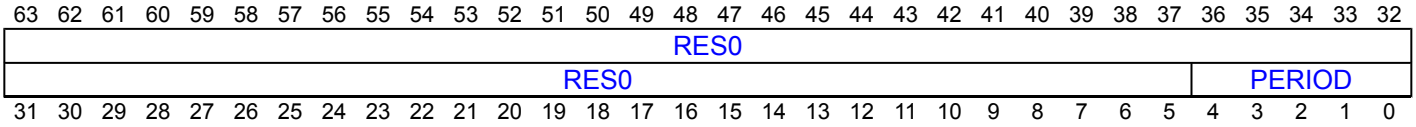
AArch64 System register TRCSYNCPR bits [31:0] are architecturally mapped to External register TRCSYNCPR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCSYNCPR are UNDEFINED.

Attributes

TRCSYNCPR is a 64-bit register.

Field descriptions



Bits [63:5]

Reserved, RES0.

PERIOD, bits [4:0]

Defines the number of bytes of trace between each periodic trace protocol synchronization request.

PERIOD	Meaning
0b00000	Trace protocol synchronization is disabled.
0b01000	Trace protocol synchronization request occurs after 2^8 bytes of trace.
0b01001	Trace protocol synchronization request occurs after 2^9 bytes of trace.
0b01010	Trace protocol synchronization request occurs after 2^{10} bytes of trace.
0b01011	Trace protocol synchronization request occurs after 2^{11} bytes of trace.
0b01100	Trace protocol synchronization request occurs after 2^{12} bytes of trace.
0b01101	Trace protocol synchronization request occurs after 2^{13} bytes of trace.
0b01110	Trace protocol synchronization request occurs after 2^{14} bytes of trace.
0b01111	Trace protocol synchronization request occurs after 2^{15} bytes of trace.
0b10000	Trace protocol synchronization request occurs after 2^{16} bytes of trace.
0b10001	Trace protocol synchronization request occurs after 2^{17} bytes of trace.
0b10010	Trace protocol synchronization request occurs after 2^{18} bytes of trace.
0b10011	Trace protocol synchronization request occurs after 2^{19} bytes of trace.
0b10100	Trace protocol synchronization request occurs after 2^{20} bytes of trace.

Other values are reserved. If a reserved value is programmed into PERIOD, then the behavior of the synchronization period counter is CONSTRAINED UNPREDICTABLE and one of the following behaviors occurs:

- No trace protocol synchronization requests are generated by this counter.
- Trace protocol synchronization requests occur at the specified period.

- Trace protocol synchronization requests occur at some other UNKNOWN period which can vary.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSYNCPR

Must be programmed if TRCIDR3.SYNCPR == 0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSYNCPR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1101	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSYNCPR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSYNCPR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSYNCPR;
```

```
MSR TRCSYNCPR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1101	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSYNCPR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSYNCPR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSYNCPR = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCTRACEIDR, Trace ID Register

The TRCTRACEIDR characteristics are:

Purpose

Sets the trace ID for instruction trace.

Configuration

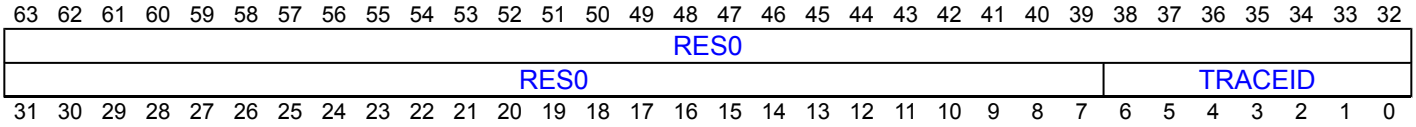
AArch64 System register TRCTRACEIDR bits [31:0] are architecturally mapped to External register TRCTRACEIDR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCTRACEIDR are UNDEFINED.

Attributes

TRCTRACEIDR is a 64-bit register.

Field descriptions



Bits [63:7]

Reserved, RES0.

TRACEID, bits [6:0]

Trace ID field. Sets the trace ID value for instruction trace. The width of the field is indicated by the value of TRCIDR5.TRACEIDSIZE. Unimplemented bits are RES0.

If an implementation supports AMBA ATB, then:

- The width of the field is 7 bits.
- Writing a reserved trace ID value does not affect behavior of the trace unit but it might cause UNPREDICTABLE behavior of the trace capture infrastructure.

See the AMBA ATB Protocol Specification for information about which ATID values are reserved.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCTRACEIDR

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCTRACEIDR

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTRACEIDR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTRACEIDR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTRACEIDR;
```

MSR TRCTRACEIDR, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTRACEIDR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTRACEIDR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTRACEIDR = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCTSCTLR, Trace Timestamp Control Register

The TRCTSCTLR characteristics are:

Purpose

Controls the insertion of global timestamps in the trace stream.

Configuration

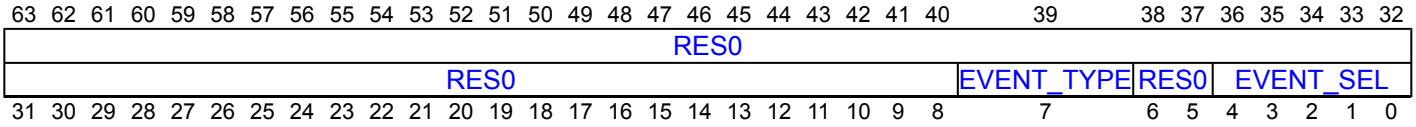
AArch64 System register TRCTSCTLR bits [31:0] are architecturally mapped to External register TRCTSCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR0.TSSIZE != 0b00000. Otherwise, direct accesses to TRCTSCTLR are UNDEFINED.

Attributes

TRCTSCTLR is a 64-bit register.

Field descriptions



Bits [63:8]

Reserved, RES0.

EVENT_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

EVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCTSCTLR.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCTSCTLR.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCTSCTLR.EVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000:

Defines the selected Resource Selector or pair of Resource Selectors. TRCTSCTLR.EVENT.TYPE controls whether TRCTSCTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCTSCTLR

Must be programmed if TRCCONFIGR.TS == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCTSCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TSSIZE !=
'00000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTSCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTSCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTSCTLR;
```

MSR TRCTSCTLR, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TSSIZE !=
'00000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTSCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTSCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTSCTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVICTLR, Trace ViewInst Main Control Register

The TRCVICTLR characteristics are:

Purpose

Controls instruction trace filtering.

Configuration

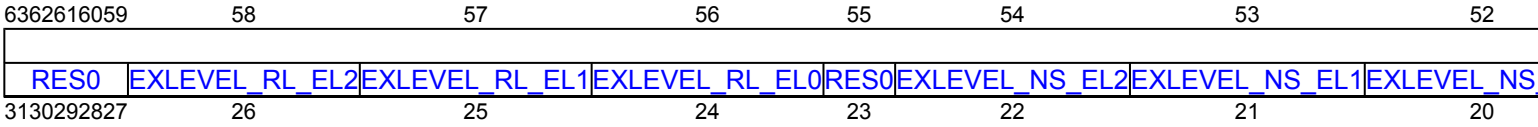
AArch64 System register TRCVICTLR bits [31:0] are architecturally mapped to External register TRCVICTLR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCVICTLR are UNDEFINED.

Attributes

TRCVICTLR is a 64-bit register.

Field descriptions



Bits [63:27]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [26]

When FEAT_RME is implemented:

Filter instruction trace for EL2 in Realm state.

EXLEVEL_RL_EL2	Meaning
0b0	When TRCVICTLR.EXLEVEL_NS_EL2 is 0 the trace unit generates instruction trace for EL2 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL2 is 1 the trace unit does not generate instruction trace for EL2 in Realm state.
0b1	When TRCVICTLR.EXLEVEL_NS_EL2 is 0 the trace unit does not generate instruction trace for EL2 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL2 is 1 the trace unit generates instruction trace for EL2 in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL1, bit [25]

When FEAT_RME is implemented:

Filter instruction trace for EL1 in Realm state.

EXLEVEL_RL_EL1	Meaning
0b0	When TRCVICTLR.EXLEVEL_NS_EL1 is 0 the trace unit generates instruction trace for EL1 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL1 is 1 the trace unit does not generate instruction trace for EL1 in Realm state.
0b1	When TRCVICTLR.EXLEVEL_NS_EL1 is 0 the trace unit does not generate instruction trace for EL1 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL1 is 1 the trace unit generates instruction trace for EL1 in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL0, bit [24]

When FEAT_RME is implemented:

Filter instruction trace for EL0 in Realm state.

EXLEVEL_RL_EL0	Meaning
0b0	When TRCVICTLR.EXLEVEL_NS_EL0 is 0 the trace unit generates instruction trace for EL0 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL0 is 1 the trace unit does not generate instruction trace for EL0 in Realm state.
0b1	When TRCVICTLR.EXLEVEL_NS_EL0 is 0 the trace unit does not generate instruction trace for EL0 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL0 is 1 the trace unit generates instruction trace for EL0 in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]

When Non-secure EL2 is implemented:

Filter instruction trace for EL2 in Non-secure state.

EXLEVEL_NS_EL2	Meaning
0b0	The trace unit generates instruction trace for EL2 in Non-secure state.
0b1	The trace unit does not generate instruction trace for EL2 in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL1, bit [21]

When Non-secure EL1 is implemented:

Filter instruction trace for EL1 in Non-secure state.

EXLEVEL_NS_EL1	Meaning
0b0	The trace unit generates instruction trace for EL1 in Non-secure state.
0b1	The trace unit does not generate instruction trace for EL1 in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL0, bit [20]

When Non-secure EL0 is implemented:

Filter instruction trace for EL0 in Non-secure state.

EXLEVEL_NS_EL0	Meaning
0b0	The trace unit generates instruction trace for EL0 in Non-secure state.
0b1	The trace unit does not generate instruction trace for EL0 in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [19]

When EL3 is implemented:

Filter instruction trace for EL3.

EXLEVEL_S_EL3	Meaning
0b0	The trace unit generates instruction trace for EL3.
0b1	The trace unit does not generate instruction trace for EL3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [18]
When Secure EL2 is implemented:

Filter instruction trace for EL2 in Secure state.

EXLEVEL_S_EL2	Meaning
0b0	The trace unit generates instruction trace for EL2 in Secure state.
0b1	The trace unit does not generate instruction trace for EL2 in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL1, bit [17]
When Secure EL1 is implemented:

Filter instruction trace for EL1 in Secure state.

EXLEVEL_S_EL1	Meaning
0b0	The trace unit generates instruction trace for EL1 in Secure state.
0b1	The trace unit does not generate instruction trace for EL1 in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [16]
When Secure EL0 is implemented:

Filter instruction trace for EL0 in Secure state.

EXLEVEL_S_EL0	Meaning
0b0	The trace unit generates instruction trace for EL0 in Secure state.
0b1	The trace unit does not generate instruction trace for EL0 in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:12]

Reserved, RES0.

TRCERR, bit [11]

When TRCIDR3.TRCERR == 1:

Controls the forced tracing of System Error exceptions.

TRCERR	Meaning
0b0	Forced tracing of System Error exceptions is disabled.
0b1	Forced tracing of System Error exceptions is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCRESET, bit [10]

Controls the forced tracing of PE Resets.

TRCRESET	Meaning
0b0	Forced tracing of PE Resets is disabled.
0b1	Forced tracing of PE Resets is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SSSTATUS, bit [9]

ViewInst start/stop function status.

SSSTATUS	Meaning
0b0	Stopped State. The ViewInst start/stop function is in the stopped state.
0b1	Started State. The ViewInst start/stop function is in the started state.

Before software enables the trace unit, it must write to this field to set the initial state of the ViewInst start/stop function. If the ViewInst start/stop function is not used then set this field to 1. Arm recommends that the value of this field is set before each trace session begins.

If the trace unit becomes disabled while a start point or stop point is still speculative, then the value of TRCVICTLR.SSSTATUS is UNKNOWN and might represent the result of a speculative start point or stop point.

If software which is running on the PE being traced disables the trace unit, either by clearing TRCPRGCTLR.EN or locking the OS Lock, Arm recommends that a DSB and an ISB instruction are executed before disabling the trace unit to prevent any start points or stop points being speculative at the point of disabling the trace unit. This procedure assumes that all start points or stop points occur before the barrier instructions are executed. The procedure does not guarantee that there are no speculative start points or stop points when disabling, although it helps minimize the probability.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES1** if all the following are true:
  - TRCIDR4.NUMACPAIRS == 0b0000.
  - TRCIDR4.NUMPC == 0b0000.

- Otherwise, access to this field is **RW**.

Bit [8]

Reserved, RES0.

EVENT_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

EVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCVICTLR.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCVICTLR.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCVICTLR.EVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

Bits[4:0]

When TRCIDR4.NUMRSPAIR != 0b0000:

EVENT_SEL, bits [4:0] of bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCVICTLR.EVENT.TYPE controls whether TRCVICTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

When TRCIDR4.NUMRSPAIR == 0b0000:

Reserved, bits [4:0] of bits [4:0]

This field is reserved:

- Bits [4:1] are RES0.

- Bit [0] is RES1.

Otherwise:

Reserved, RES0.

Accessing TRCVICTLR

Must be programmed.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVICTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRCVICTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVICTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVICTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVICTLR;
```

```
MSR TRCVICTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRCVICTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVICTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVICTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVICTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVIIECTLR, Trace ViewInst Include/Exclude Control Register

The TRCVIIECTLR characteristics are:

Purpose

Use this to select, or read, the Address Range Comparators for the ViewInst include/exclude function.

Configuration

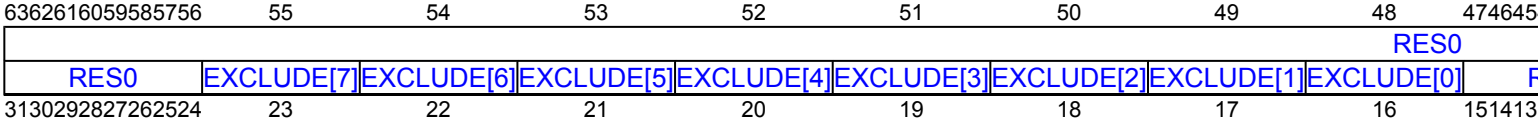
AArch64 System register TRCVIIECTLR bits [31:0] are architecturally mapped to External register TRCVIIECTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMACPAIRS) > 0. Otherwise, direct accesses to TRCVIIECTLR are UNDEFINED.

Attributes

TRCVIIECTLR is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

EXCLUDE[<m>], bit [m+16], for m = 7 to 0

Exclude Address Range Comparator <m>. Selects whether Address Range Comparator <m> is in use with the ViewInst exclude function.

EXCLUDE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines, is not selected for the ViewInst exclude function.
0b1	The address range that Address Range Comparator <m> defines, is selected for the ViewInst exclude function.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Bits [15:8]

Reserved, RES0.

INCLUDE[<m>], bit [m], for m = 7 to 0

Include Address Range Comparator <m>.

Selects whether Address Range Comparator <m> is in use with the ViewInst include function.

Selecting no comparators for the ViewInst include function indicates that all instructions are included by default.

The ViewInst exclude function then indicates which ranges are excluded.

INCLUDE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines, is not selected for the ViewInst include function.
0b1	The address range that Address Range Comparator <m> defines, is selected for the ViewInst include function.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCVIIECTLR

Must be programmed if TRCIDR4.NUMACPAIRS > 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVIIECTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) &&
UInt(TRCIDR4.NUMACPAIRS) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIIECTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIIECTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIIECTLR;
```

```
MSR TRCVIIECTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) &&
UInt(TRCIDR4.NUMACPAIRS) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIIECTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIIECTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIIECTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVIPCSSCTLR, Trace ViewInst Start/Stop PE Comparator Control Register

The TRCVIPCSSCTLR characteristics are:

Purpose

Use this to select, or read, which PE Comparator Inputs can control the ViewInst start/stop function.

Configuration

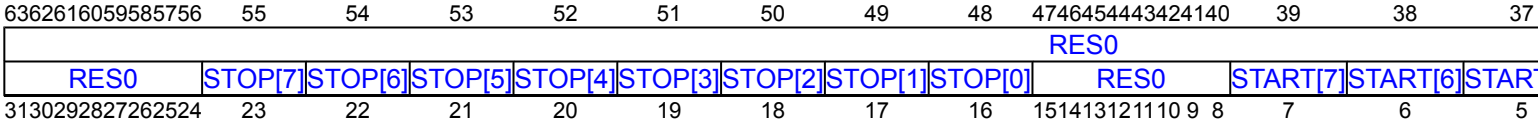
AArch64 System register TRCVIPCSSCTLR bits [31:0] are architecturally mapped to External register TRCVIPCSSCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMPC) > 0. Otherwise, direct accesses to TRCVIPCSSCTLR are UNDEFINED.

Attributes

TRCVIPCSSCTLR is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

STOP[<m>], bit [m+16], for m = 7 to 0

Selects whether PE Comparator Input <m> is in use with the ViewInst start/stop function for the purpose of stopping trace.

STOP[<m>]	Meaning
0b0	The PE Comparator Input <m> is not selected as a stop resource.
0b1	The PE Comparator Input <m> is selected as a stop resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMPC), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Bits [15:8]

Reserved, RES0.

START[<m>], bit [m], for m = 7 to 0

Selects whether PE Comparator Input <m> is in use with the ViewInst start/stop function for the purpose of starting trace.

START[<m>]	Meaning
0b0	The PE Comparator Input <m> is not selected as a start resource.
0b1	The PE Comparator Input <m> is selected as a start resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMPC), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCVIPCSSCTLR

Must be programmed if TRCIDR4.NUMPC != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVIPCSSCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMPC) >
0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIPCSSCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIPCSSCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIPCSSCTLR;
```

MSR TRCVIPCSSCTLR, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMPC) >
0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIPCSSCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIPCSSCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIPCSSCTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVISSCTLR, Trace ViewInst Start/Stop Control Register

The TRCVISSCTLR characteristics are:

Purpose

Use this to select, or read, the Single Address Comparators for the ViewInst start/stop function.

Configuration

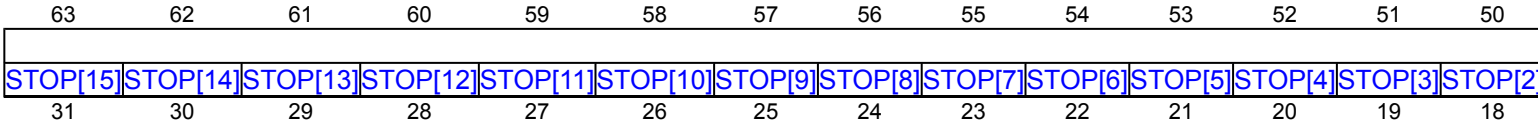
AArch64 System register TRCVISSCTLR bits [31:0] are architecturally mapped to External register TRCVISSCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMACPAIRS) > 0. Otherwise, direct accesses to TRCVISSCTLR are UNDEFINED.

Attributes

TRCVISSCTLR is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

STOP[<m>], bit [m+16], for m = 15 to 0

Selects whether Single Address Comparator <m> is used with the ViewInst start/stop function for the purpose of stopping trace.

STOP[<m>]	Meaning
0b0	The Single Address Comparator <m> is not selected as a stop resource.
0b1	The Single Address Comparator <m> is selected as a stop resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS) * 2, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

START[<m>], bit [m], for m = 15 to 0

Selects whether Single Address Comparator <m> is used with the ViewInst start/stop function for the purpose of starting trace.

START[<m>]	Meaning
0b0	The Single Address Comparator <m> is not selected as a start resource.
0b1	The Single Address Comparator <m> is selected as a start resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS) * 2, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCVISSCTLR

Must be programmed if TRCIDR4.NUMACPAIRS > `0b0000`.

For any 2 comparators selected for the ViewInst start/stop function, the comparator containing the lower address must be a lower numbered comparator.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVISSCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) &&
UInt(TRCIDR4.NUMACPAIRS) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVISSCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVISSCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVISSCTLR;
```

```
MSR TRCVISSCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) &&
UInt(TRCIDR4.NUMACPAIRS) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVISSCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVISSCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVISSCTLR = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCCTLR0, Trace Virtual Context Identifier Comparator Control Register 0

The TRCVMIDCCTLR0 characteristics are:

Purpose

Virtual Context Identifier Comparator mask values for the TRCVMIDCVR<n> registers, where n=0-3.

Configuration

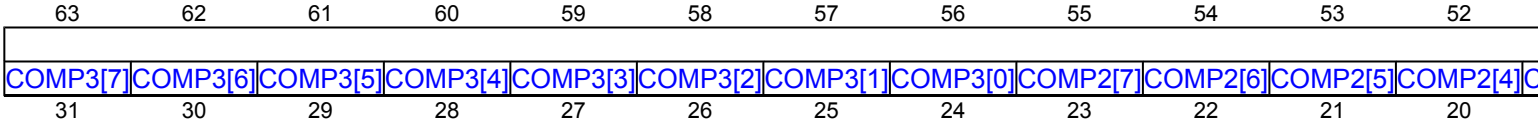
AArch64 System register TRCVMIDCCTLR0 bits [31:0] are architecturally mapped to External register TRCVMIDCCTLR0[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, UInt(TRCIDR4.NUMVMIDC) > 0x0, and UInt(TRCIDR2.VMIDSIZE) > 0. Otherwise, direct accesses to TRCVMIDCCTLR0 are UNDEFINED.

Attributes

TRCVMIDCCTLR0 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

COMP3[<m>], bit [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 3:

TRCVMIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR3. Each bit in this field corresponds to a byte in TRCVMIDCVR3.

COMP3[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP2[<m>], bit [m+16], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 2:

TRCVMIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR2. Each bit in this field corresponds to a byte in TRCVMIDCVR2.

COMP2[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP1[<m>], bit [m+8], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 1:

TRCVMIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR1. Each bit in this field corresponds to a byte in TRCVMIDCVR1.

COMP1[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP0[<m>], bit [m], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 0:

TRCVMIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR0. Each bit in this field corresponds to a byte in TRCVMIDCVR0.

COMP0[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Accessing TRCVMIDCCTLR0

If software uses the TRCVMIDCVR<n> registers, where n=0-3, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCVMIDCVR<n> to `0x00`.

If any bit is 1 and the relevant byte in TRCVMIDCVR<n> is not `0x00`, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVMIDCCTLR0
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMVMIDC)
> 0x0 && UInt(TRCIDR2.VMIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR0;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR0;
```

```
MSR TRCVMIDCCTLR0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMVMIDC)
> 0x0 && UInt(TRCIDR2.VMIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR0 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCCTLR1, Trace Virtual Context Identifier Comparator Control Register 1

The TRCVMIDCCTLR1 characteristics are:

Purpose

Virtual Context Identifier Comparator mask values for the TRCVMIDCVR<n> registers, where n=4-7.

Configuration

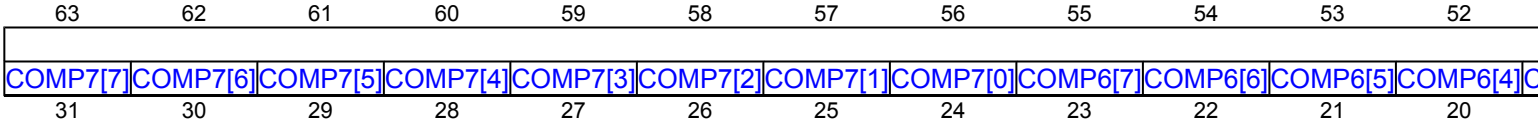
AArch64 System register TRCVMIDCCTLR1 bits [31:0] are architecturally mapped to External register TRCVMIDCCTLR1[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, UInt(TRCIDR4.NUMVMIDC) > 0x4, and UInt(TRCIDR2.VMIDSIZE) > 0. Otherwise, direct accesses to TRCVMIDCCTLR1 are UNDEFINED.

Attributes

TRCVMIDCCTLR1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

COMP7[<m>], bit [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 7:

TRCVMIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR7. Each bit in this field corresponds to a byte in TRCVMIDCVR7.

COMP7[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP6[<m>], bit [m+16], for m = 7 to 0
When UInt(TRCIDR4.NUMVMIDC) > 6:

TRCVMIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR6. Each bit in this field corresponds to a byte in TRCVMIDCVR6.

COMP6[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR6[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR6[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP5[<m>], bit [m+8], for m = 7 to 0
When UInt(TRCIDR4.NUMVMIDC) > 5:

TRCVMIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR5. Each bit in this field corresponds to a byte in TRCVMIDCVR5.

COMP5[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR5[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR5[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP4[<m>], bit [m], for m = 7 to 0
When UInt(TRCIDR4.NUMVMIDC) > 4:

TRCVMIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR4. Each bit in this field corresponds to a byte in TRCVMIDCVR4.

COMP4[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR4[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR4[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Accessing TRCVMIDCCTLR1

If software uses the TRCVMIDCVR<n> registers, where n=4-7, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCVMIDCVR<n> to `0x00`.

If any bit is 1 and the relevant byte in TRCVMIDCVR<n> is not `0x00`, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVMIDCCTLR1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMVMIDC)
> 0x4 && UInt(TRCIDR2.VMIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR1;
```

MSR TRCVMIDCCTLR1, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMVMIDC)
> 0x4 && UInt(TRCIDR2.VMIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCVR<n>, Trace Virtual Context Identifier Comparator Value Register <n>, n = 0 - 7

The TRCVMIDCVR<n> characteristics are:

Purpose

Contains the Virtual Context Identifier Comparator value.

Configuration

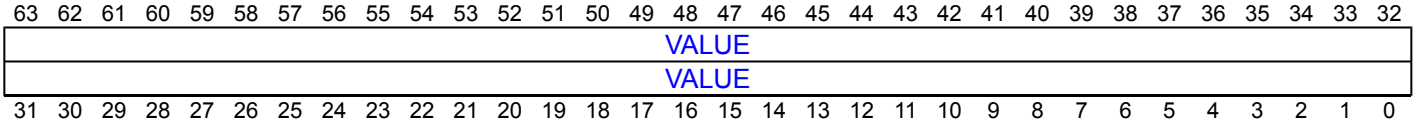
AArch64 System register TRCVMIDCVR<n> bits [63:0] are architecturally mapped to External register TRCVMIDCVR<n>[63:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMVMIDC) > n. Otherwise, direct accesses to TRCVMIDCVR<n> are UNDEFINED.

Attributes

TRCVMIDCVR<n> is a 64-bit register.

Field descriptions



VALUE, bits [63:0]

Virtual context identifier value. The width of this field is indicated by TRCIDR2.VMIDSIZE. Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Virtual context identifier is zero until the PE updates the Virtual context identifier .

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVMIDCVR<n>

Must be programmed if any of the following are true:

- TRCRSCTLR<a>.GROUP == 0b0111 and TRCRSCTLR<a>.VMID[n] == 1.
- TRCACATR<a>.CONTEXTTYPE == 0b10 or 0b11 and TRCACATR<a>.CONTEXT == n.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVMIDCVR<m> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	m[2:0]:0b0	0b001

```
integer m = UInt(CRm<3:1>);

if m >= NUM_TRACE_VIRTUAL_CONTEXT_IDENTIFIER_COMPARATORS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCVR[m];
```

```
MSR TRCVMIDCVR<m>, <Xt> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	m[2:0]:0b0	0b001

```
integer m = UInt(CRm<3:1>);

if m >= NUM_TRACE_VIRTUAL_CONTEXT_IDENTIFIER_COMPARATORS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
EDSCR2.TTA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCVR[m] = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRFCR_EL1, Trace Filter Control Register (EL1)

The TRFCR_EL1 characteristics are:

Purpose

Provides EL1 controls for Trace.

Configuration

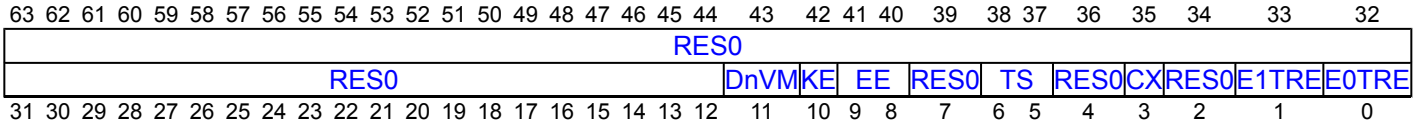
AArch64 System register TRFCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TRFCR[31:0].

This register is present only when FEAT_TRF is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TRFCR_EL1 are UNDEFINED.

Attributes

TRFCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:12]

Reserved, RES0.

DnVM, bit [11]

When FEAT_TRBEv1p1 is implemented and FEAT_NV is implemented:

Reserved for software use in nested virtualization. See also TRFCR_EL2.DnVM.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

KE, bit [10]

When FEAT_TRBE_EXC is implemented:

Kernel exception enable for TRBE Profiling exceptions taken to EL1.

KE	Meaning
0b0	TRBE Profiling exceptions taken to EL1 are always masked at EL1.
0b1	Enabled TRBE Profiling exceptions taken to EL1 are masked at EL1 when PSTATE.PM is 1 and unmasked when PSTATE.PM is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bits [9:8]

When FEAT_TRBE_EXC is implemented:

Exception Enable.

EE	Meaning	Applies when
0b00	Disabled. TRBE Profiling exceptions for EL1 are disabled. All of the following apply: • Unless enabled by a higher Exception level, TRBE Profiling exceptions are not generated. • TRBSR_EL1.IRQ drives the interrupt request signal **TRBIRQ**. • Accesses to TRBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1.	
0b01	Reserved for software use in nested virtualization. Behaves as 0b00 for the purpose of controlling the TRBE Profiling exception and interrupt request signal **TRBIRQ**, and as 0b11 for the purpose of accesses to TRBSR_EL1.	When FEAT_NV is implemented
0b10	Reserved for software use in nested virtualization. Behaves as 0b11 for the purposes of controlling the TRBE Profiling exception and interrupt request signal **TRBIRQ**, and accesses to TRBSR_EL1.	When FEAT_NV is implemented
0b11	Enabled. TRBE Profiling exceptions for EL1 are enabled, as follows: • All trace buffer management events are recorded in TRBSR_EL1, unless they are configured to be recorded in TRBSR_EL3 by MDCR_EL3.TRBEE or TRBSR_EL2 by TRFCR_EL2.EE. • TRBE Profiling exceptions are generated and taken to EL1 when unmasked and TRBSR_EL1.IRQ is 1, unless the Effective value of HCR_EL2.TGE is 1, in which case the exception is taken to EL2. • The interrupt request signal **TRBIRQ** is not asserted.	

For more information on the values reserved for software use in nested virtualization, see TRFCR_EL2.EE.

If the Effective value of TRFCR_EL2.EE is 0b00, then the Effective value of TRFCR_EL1.EE is 0b00.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control. Controls which timebase is used for trace timestamps.

TS	Meaning	Applies when
0b00	Reserved for software use in nested virtualization. Behaves as 0b01. See also TRFCR_EL2.TS,	When FEAT_NV2p1 is implemented
0b01	Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF_EL2.	
0b10	Guest physical timestamp. The traced timestamp is the physical counter value minus a physical offset. If any of the following are true, then the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • SCR_EL3.ECVEn == 0. • CNTHCTL_EL2.ECV == 0. • FEAT_ECV_POFF is not implemented.	When FEAT_ECV is implemented
0b11	Physical timestamp. The traced timestamp is the physical counter value.	

All other values are reserved.

If any of the following are true, then this field is ignored by the PE:

- EL2 is implemented and TRFCR_EL2.TS is not 0b00.
- SelfHostedTraceEnabled() == FALSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [4]

Reserved, RES0.

CX, bit [3]

When FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization. See also TRFCR_EL2.CX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [2]

Reserved, RES0.

E1TRE, bit [1]

EL1 Trace Enable.

E1TRE	Meaning
0b0	Trace is prohibited at EL1.
0b1	Trace is allowed at EL1.

If any of the following are true, then this field is ignored by the PE:

- SelfHostedTraceEnabled() == FALSE.
- The Effective value of HCR_EL2.TGE is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0TRE, bit [0]

EL0 Trace Enable.

E0TRE	Meaning
0b0	Trace is prohibited at EL0.
0b1	Trace is allowed at EL0.

If any of the following are true, then this field is ignored by the PE:

- SelfHostedTraceEnabled() == FALSE.
- The Effective value of HCR_EL2.TGE is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing TRFCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRFCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x880];
    else
        X[t, 64] = TRFCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TRFCR_EL2;
    else
        X[t, 64] = TRFCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRFCR_EL1;
```

```
MSR TRFCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x880] = X[t, 64];
    else
        TRFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TRFCR_EL2 = X[t, 64];
    else
        TRFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRFCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, TRFCR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x880];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TRFCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TRFCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR TRFCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x880] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRFCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TRFCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRFCR_EL2, Trace Filter Control Register (EL2)

The TRFCR_EL2 characteristics are:

Purpose

Provides EL2 controls for Trace.

Configuration

AArch64 System register TRFCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HTRFCR[31:0].

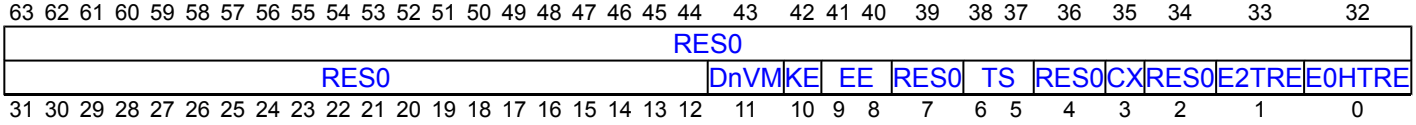
This register is present only when FEAT_TRF is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TRFCR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

TRFCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:12]

Reserved, RES0.

DnVM, bit [11]

When FEAT_TRBEv1p1 is implemented:

Disable use of physical address trace buffer pointers.

DnVM	Meaning
0b0	Use of physical address trace buffer pointers is permitted.
0b1	Use of physical address trace buffer pointers is disabled. The PE behaves as if TRBLIMITR_EL1.nVM is 0.

If EL2 is disabled in the owning Security state, or the trace buffer owning Exception level is EL2, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

KE, bit [10]

When FEAT_TRBE_EXC is implemented:

Kernel exception enable for TRBE Profiling exceptions taken to EL2.

KE	Meaning
0b0	TRBE Profiling exceptions taken to EL2 are always masked at EL2.
0b1	Enabled TRBE Profiling exceptions taken to EL2 are masked at EL2 when PSTATE.PM is 1 and unmasked when PSTATE.PM is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bits [9:8]

When FEAT_TRBE_EXC is implemented:

Exception Enable.

EE	Meaning
0b00	Disabled. TRBE Profiling exceptions for EL2 and EL1 are disabled. All of the following apply: • No trace buffer management events are recorded in TRBSR_EL2. • Unless enabled by a higher Exception level, TRBE Profiling exceptions are not generated. • TRBSR_EL1.IRQ drives the interrupt request signal **TRBIRQ**. • Accesses to TRBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1 and accesses to TRBSR_EL1 at EL2 ignore the value of HCR_EL2.E2H.
0b01	Delegated. TRBE Profiling exceptions for EL2 are disabled, but might be enabled for EL1 by TRFCR_EL1.EE. All of the following apply: • No trace buffer management events are recorded in TRBSR_EL2. • TRBSR_EL2.IRQ is ignored and TRBE Profiling exceptions are not taken to EL2, other than for the case when the Effective value of HCR_EL2.TGE is 1.
0b10	Enabled. TRBE Profiling exceptions for EL2 are enabled for trace buffer management events targeting EL2, as follows: • Trace buffer management events due to a fault on a write to the trace buffer that would generate a Data Abort exception taken to EL2 if generated by a store instruction executed at the owning Exception level are recorded in TRBSR_EL2, unless they are configured to be recorded in TRBSR_EL3 by MDCR_EL3.TRBEE. If the trace buffer owning Exception level is EL2, then this means any fault on a write to the trace buffer. If the trace buffer owning Exception level is EL1, then this means any of the following faults on a write to the trace buffer: ◦ Stage 2 faults. ◦ If HCR_EL2.TEA is 1, synchronous External aborts. ◦ If HCR_EL2.GPF is 1, Granule Protection Faults (GPFs). • Trace buffer management events due to Granule Protection Check faults other than GPFs on a write to the trace buffer are recorded in TRBSR_EL2, unless they are configured to be recorded in TRBSR_EL3 by MDCR_EL3.TRBEE. • TRBE Profiling exceptions are generated and taken to EL2 when unmasked and TRBSR_EL2.IRQ is 1.
0b11	Trap all. TRBE Profiling exceptions for EL2 are enabled for all trace buffer management events, as follows: • All trace buffer management events are recorded in TRBSR_EL2, unless they are configured to be recorded in TRBSR_EL3 by MDCR_EL3.TRBEE. • TRBE Profiling exceptions are generated and taken to EL2 when unmasked and TRBSR_EL2.IRQ is 1.

If the Effective value of MDCR_EL3.TRBEE is 0b00, then the Effective value of TRFCR_EL2.EE is 0b00. Otherwise, if EL2 is not implemented or the Effective value of SCR_EL3.{NS, EEL2} is {0, 0}, then the Effective value of TRFCR_EL2.EE is 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control. Controls which timebase is used for trace timestamps.

TS	Meaning	Applies when
0b00	Timestamp controlled by TRFCR_EL1.TS or TRFCR.TS.	
0b01	Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF_EL2.	
0b10	Guest physical timestamp. The traced timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • SCR_EL3.ECVEn == 0. • CNTHCTL_EL2.ECV == 0. • FEAT_ECV_POFF is not implemented.	When FEAT_ECV is implemented
0b11	Physical timestamp. The traced timestamp is the physical counter value.	

If SelfHostedTraceEnabled() == FALSE, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

Bit [4]

Reserved, RES0.

CX, bit [3]

CONTEXTIDR_EL2 and VMID trace enable.

CX	Meaning
0b0	CONTEXTIDR_EL2 and VMID trace prohibited.
0b1	CONTEXTIDR_EL2 and VMID trace allowed.

If SelfHostedTraceEnabled() == FALSE, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [2]

Reserved, RES0.

E2TRE, bit [1]

EL2 Trace Enable.

E2TRE	Meaning
0b0	Trace is prohibited at EL2.
0b1	Trace is allowed at EL2.

If SelfHostedTraceEnabled() == FALSE, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0HTRE, bit [0]

EL0 Trace Enable.

E0HTRE	Meaning
0b0	Trace is prohibited at EL0.
0b1	Trace is allowed at EL0.

If any of the following are true, then this field is ignored by the PE:

- SelfHostedTraceEnabled() == FALSE.
- The Effective value of HCR_EL2.TGE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing TRFCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRFCR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TRFCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRFCR_EL2;
```

```
MSR TRFCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRFCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRFCR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, TRFCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x880];
    else
        X[t, 64] = TRFCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TRFCR_EL2;
    else
        X[t, 64] = TRFCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRFCR_EL1;
```

When FEAT_VHE is implemented

```
MSR TRFCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HDFGWTR_EL2.TRFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x880] = X[t, 64];
    else
        TRFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TRFCR_EL2 = X[t, 64];
    else
        TRFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRFCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR0_EL1, Translation Table Base Register 0 (EL1)

The TTBR0_EL1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the EL1&0 translation regime, and other information for this translation regime.

Configuration

AArch64 System register TTBR0_EL1 bits [63:0] are architecturally mapped to AArch32 System register TTBR0[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TTBR0_EL1 are UNDEFINED.

TTBR0_EL1 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

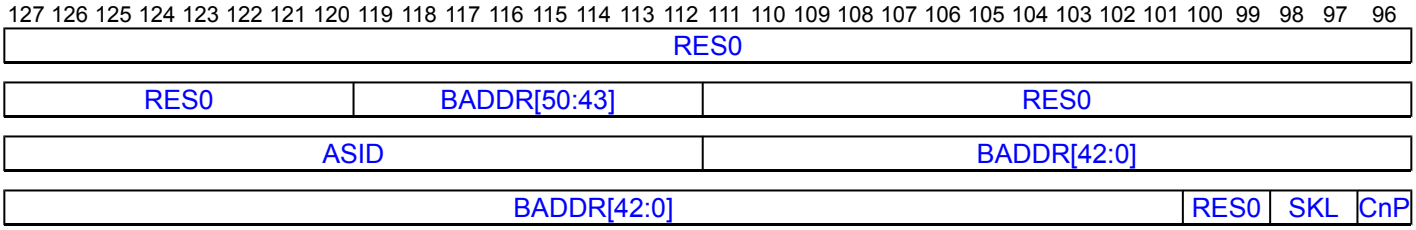
Attributes

TTBR0_EL1 is a:

- 128-bit register when FEAT_D128 is implemented and TCR2_EL1.D128 == 1
- 64-bit register when FEAT_D128 is not implemented or TCR2_EL1.D128 == 0

Field descriptions

When FEAT_D128 is implemented and TCR2_EL1.D128 == 1:



Bits [127:88]

Reserved, RES0.

BADDR, bits [87:80, 47:5]

Translation table base address:

- Bits A[55:x] of the stage 1 translation table base address bits are in register bits[87:80, 47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The BADDR field is split as follows:

- BADDR[50:43] is TTBR0_EL1[87:80].
- BADDR[42:0] is TTBR0_EL1[47:5].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [79:64]

Reserved, RES0.

ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level associated with translation table walks using TTBR0_EL1.

This determines the number of levels to be skipped from the regular start level of the stage 1 EL1&0 translation table walks using TTBR0_EL1.

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR0_EL1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR0_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.
0b1	The translation table entries pointed to by TTBR0_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR0_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

This bit is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR0_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

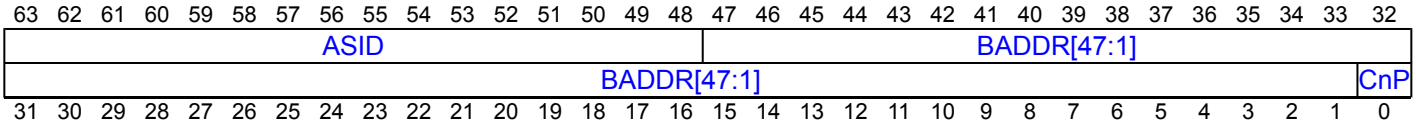
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When FEAT_D128 is not implemented or TCR2_EL1.D128 == 0:



ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL1.T0SZ, the translation stage, and the translation granule size.

If the value of TCR_EL1.IPS is not `0b110`, then:

- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits A[51:48] of the stage 1 translation table base address are `0b0000`.

The BADDR field represents a 52-bit address if one of the following applies:

- FEAT_LPA is implemented, the 64KB granule size is in use, and the value of TCR_EL1.IPS is `0b110`.
- FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of TCR_EL1.DS is 1.
- FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of TCR2_EL1.D128 is 0.

When TTBR0_EL1.BADDR represents a 52-bit addresses, all of the following apply:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- The smallest permitted value of x is 6.
- When x>6, register bits[(x-1):6] are RES0.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

For the 64KB granule, if FEAT_LPA is not implemented, and the value of TCR_EL1.IPS is `0b110`, one the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR0_EL1, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]
When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1.

CnP	Meaning
`0b0`	The translation table entries pointed to by TTBR0_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR0_EL1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR0_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.
`0b1`	The translation table entries pointed to by TTBR0_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR0_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

This bit is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR0_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TTBR0_EL1 or TTBR0_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TTBR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x200];
    else
        X[t, 64] = TTBR0_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR0_EL2<63:0>;
    else
        X[t, 64] = TTBR0_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR0_EL1<63:0>;
```

```
MSR TTBR0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x200] = X[t, 64];
    else
        TTBR0_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR0_EL2<63:0> = X[t, 64];
    else
        TTBR0_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1<63:0> = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, TTBR0_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x200];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR0_EL1<63:0>;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR0_EL1<63:0>;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR TTBR0_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x200] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR0_EL1<63:0> = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TTBR0_EL1<63:0> = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_D128 is implemented

```
MRRS <Xt>, <Xt+1>, TTBR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, t2, 128] = NVMem[0x200, 128];
    else
        X[t, t2, 128] = TTBR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        X[t, t2, 128] = TTBR0_EL2;
    else
        X[t, t2, 128] = TTBR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = TTBR0_EL1;
```

When FEAT_D128 is implemented

```
MSRR TTBR0_EL1, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x200, 128] = X[t, t2, 128];
    else
        TTBR0_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        TTBR0_EL2<127:0> = X[t, t2, 128];
    else
        TTBR0_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1<127:0> = X[t, t2, 128];
```

When FEAT_D128 is implemented and FEAT_VHE is implemented

```
MRRS <Xt>, <Xt+1>, TTBR0_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, t2, 128] = NVMem[0x200, 128];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            X[t, t2, 128] = TTBR0_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, t2, 128] = TTBR0_EL1;
    else
        UNDEFINED;
```

When FEAT_D128 is implemented and FEAT_VHE is implemented

```
MSRR TTBR0_EL12, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x200, 128] = X[t, t2, 128];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            TTBR0_EL1<127:0> = X[t, t2, 128];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TTBR0_EL1<127:0> = X[t, t2, 128];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR0_EL2, Translation Table Base Register 0 (EL2)

The TTBR0_EL2 characteristics are:

Purpose

When the Effective value of HCR_EL2.E2H is not 1, holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL2 translation regime, and other information for this translation regime.

When the Effective value of HCR_EL2.E2H is 1, holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the EL2&0 translation regime, and other information for this translation regime.

Configuration

AArch64 System register TTBR0_EL2 bits [47:0] are architecturally mapped to AArch32 System register HTTBR[47:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TTBR0_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

TTBR0_EL2 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

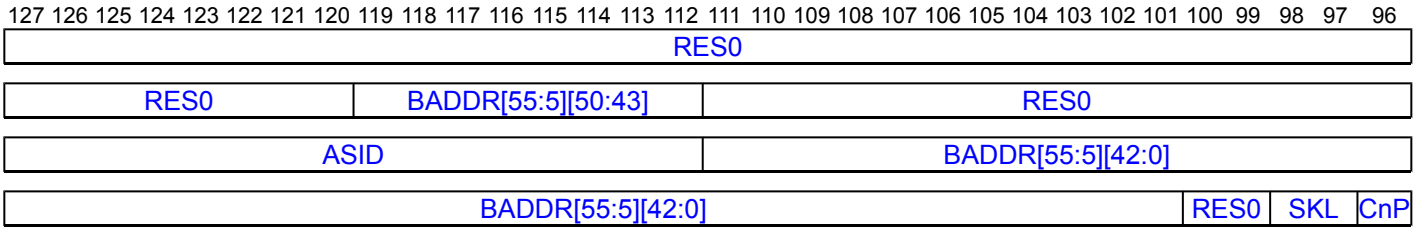
Attributes

TTBR0_EL2 is a:

- 128-bit register when FEAT_D128 is implemented, TCR2_EL2.D128 == 1, and ELIsInHost(EL2)
- 64-bit register when FEAT_D128 is not implemented or TCR2_EL2.D128 == 0

Field descriptions

When FEAT_D128 is implemented, TCR2_EL2.D128 == 1, and ELIsInHost(EL2):



Bits [127:88]

Reserved, RES0.

BADDR[55:5], bits [87:80, 47:5]

Translation table base address:

- Bits A[55:x] of the stage 1 translation table base address bits are in register bits[87:80, 47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The BADDR[55:5] field is split as follows:

- BADDR[55:5][50:43] is TTBR0_EL2[87:80].
- BADDR[55:5][42:0] is TTBR0_EL2[47:5].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [79:64]

Reserved, RES0.

ASID, bits [63:48]

When FEAT_VHE is implemented:

When the Effective value of HCR_EL2.E2H is 1, it holds an ASID for the translation table base address. The TCR_EL2.A1 field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level associated with translation table walks using TTBR0_EL2.

This determines the number of levels to be skipped from the regular start level of the stage 1 EL2&0 translation table walks using TTBR0_EL2.

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL2 for the current translation regime, and ASID if applicable, are permitted to differ from corresponding entries for TTBR0_EL2 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR0_EL2.CnP on those other PEs. • When the current translation regime is the EL2&0 regime, the value of the current ASID.
0b1	The translation table entries pointed to by TTBR0_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR0_EL2. • The translation tables relate to the same translation regime. • If that translation regime is the EL2&0 regime, the ASID is the same as the current ASID.

This bit is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

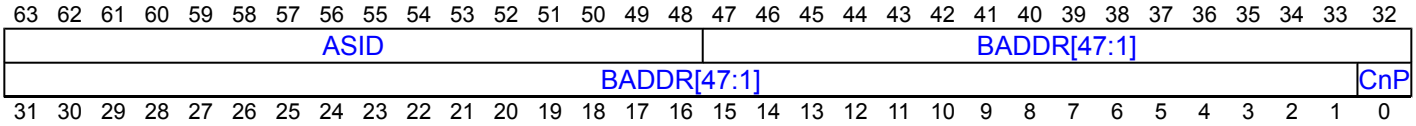
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When FEAT_D128 is not implemented or TCR2_EL2.D128 == 0:



ASID, bits [63:48]

When FEAT_VHE is implemented:

When the Effective value of HCR_EL2.E2H is not 1, this field is RES0.

When the Effective value of HCR_EL2.E2H is 1, it holds an ASID for the translation table base address. The TCR_EL2.A1 field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL2.T0SZ, the translation stage, and the translation granule size.

If the value of TCR_EL2.{I}PS is not `0b110`, then:

- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits A[51:48] of the stage 1 translation table base address are `0b0000`.

The BADDR field represents a 52-bit address if one of the following applies:

- FEAT_LPA is implemented, the 64KB granule size is in use, and the value of TCR_EL2.{I}PS is `0b110`.
- FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of TCR_EL2.DS is 1.
- FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of TCR2_EL2.D128 is 0.

When TTBR0_EL2.BADDR represents a 52-bit addresses, all of the following apply:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- The smallest permitted value of x is 6.
- When x>6, register bits[(x-1):6] are RES0.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

The OA size specified by TCR_EL2.{I}PS is determined as follows:

- The value of TCR_EL2.PS when the Effective value of HCR_EL2.E2H is not 1.
- The value of TCR_EL2.IPS when the Effective value of HCR_EL2.E2H is 1.

For the 64KB granule, if FEAT_LPA is not implemented, and the value of TCR_EL2.{I}PS is `0b110`, one the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR0_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL2 for the current translation regime, and ASID if applicable, are permitted to differ from corresponding entries for TTBR0_EL2 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR0_EL2.CnP on those other PEs. • When the current translation regime is the EL2&0 regime, the value of the current ASID.
0b1	The translation table entries pointed to by TTBR0_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR0_EL2. • The translation tables relate to the same translation regime. • If that translation regime is the EL2&0 regime, the ASID is the same as the current ASID.

This bit is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TTBR0_EL2 or TTBR0_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TTBR0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TTBR0_EL2<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR0_EL2<63:0>;
```

MSR TTBR0_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TTBR0_EL2<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR0_EL2<63:0> = X[t, 64];
```

MRS <Xt>, TTBR0_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x200];
    else
        X[t, 64] = TTBR0_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR0_EL2<63:0>;
    else
        X[t, 64] = TTBR0_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR0_EL1<63:0>;
```

MSR TTBR0_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x200] = X[t, 64];
    else
        TTBR0_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR0_EL2<63:0> = X[t, 64];
    else
        TTBR0_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1<63:0> = X[t, 64];
```

When FEAT_D128 is implemented

```
MRRS <Xt>, <Xt+1>, TTBR0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = TTBR0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = TTBR0_EL2;
```

When FEAT_D128 is implemented

```
MSRR TTBR0_EL2, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        TTBR0_EL2<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR0_EL2<127:0> = X[t, t2, 128];
```

When FEAT_D128 is implemented

```
MRRS <Xt>, <Xt+1>, TTBR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, t2, 128] = NVMem[0x200, 128];
    else
        X[t, t2, 128] = TTBR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        X[t, t2, 128] = TTBR0_EL2;
    else
        X[t, t2, 128] = TTBR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = TTBR0_EL1;
```

When FEAT_D128 is implemented

```
MSRR TTBR0_EL1, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x200, 128] = X[t, t2, 128];
    else
        TTBR0_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        TTBR0_EL2<127:0> = X[t, t2, 128];
    else
        TTBR0_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1<127:0> = X[t, t2, 128];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR0_EL3, Translation Table Base Register 0 (EL3)

The TTBR0_EL3 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL3 translation regime, and other information for this translation regime.

Configuration

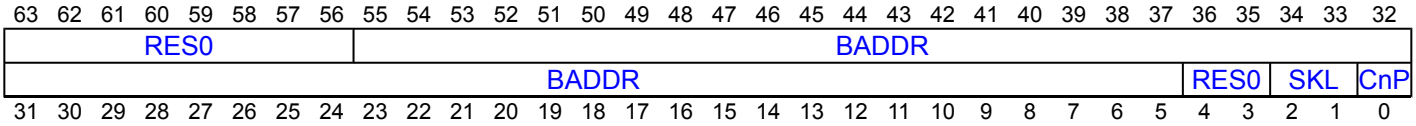
This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TTBR0_EL3 are UNDEFINED.

Attributes

TTBR0_EL3 is a 64-bit register.

Field descriptions

When FEAT_D128 is implemented and TCR_EL3.D128 == 1:



Bits [63:56]

Reserved, RES0.

BADDR, bits [55:5]

- Bits A[55:x] of the stage 1 translation table base address bits are in register bits[55:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level associated with translation table walks using TTBR0_EL3.

This determines the number of levels to be skipped from the regular start level of the stage 1 EL3 translation table walks using TTBR0_EL3.

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL3 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL3, for the current translation regime, are permitted to differ from corresponding entries for TTBR0_EL3 for other PEs in the Inner Shareable domain. This is not affected by the value of TTBR0_EL3.CnP on those other PEs.
0b1	The translation table entries pointed to by TTBR0_EL3 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1 and the translation table entries are pointed to by TTBR0_EL3.

This bit is permitted to be cached in a TLB.

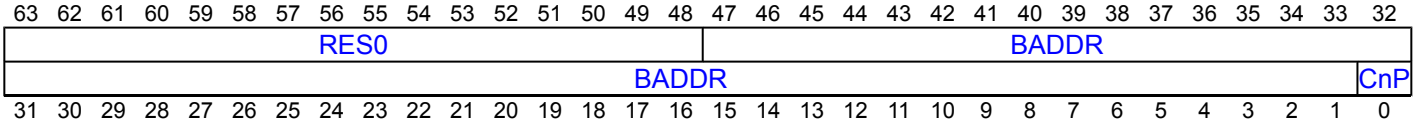
Note

If the value of the TTBR0_EL3.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL3s do not point to the same translation table entries the results of translations using TTBR0_EL3 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is not implemented or TCR_EL3.D128 == 0:



Bits [63:48]

Reserved, RES0.

BADDR, bits [47:1]

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL3.T0SZ, the translation stage, and the translation granule size.

If the value of TCR_EL3.PS is not 0b110, then:

- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits A[51:48] of the stage 1 translation table base address are 0b0000.

The BADDR field represents a 52-bit address if one of the following applies:

- FEAT_LPA is implemented, the 64KB granule size is in use, and the value of TCR_EL3.PS is 0b110.
- FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of TCR_EL3.DS is 1.
- FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of TCR_EL3.D128 is 0.

When TTBR0_EL3.BADDR represents a 52-bit addresses, all of the following apply:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- The smallest permitted value of x is 6.
- When x>6, register bits[(x-1):6] are RES0.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

For the 64KB granule, if FEAT_LPA is not implemented, and the value of TCR_EL3.PS is 0b110, one the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR0_EL3, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]
When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL3 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL3, for the current translation regime, are permitted to differ from corresponding entries for TTBR0_EL3 for other PEs in the Inner Shareable domain. This is not affected by the value of TTBR0_EL3.CnP on those other PEs.
0b1	The translation table entries pointed to by TTBR0_EL3 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1 and the translation table entries are pointed to by TTBR0_EL3.

This bit is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL3.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL3s do not point to the same translation table entries the results of translations using TTBR0_EL3 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TTBR0_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR0_EL3;
```

```
MSR TTBR0_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.TTBR0_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TTBR0_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR1_EL1, Translation Table Base Register 1 (EL1)

The TTBR1_EL1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the EL1&0 stage 1 translation regime, and other information for this translation regime.

Configuration

AArch64 System register TTBR1_EL1 bits [63:0] are architecturally mapped to AArch32 System register TTBR1[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TTBR1_EL1 are UNDEFINED.

TTBR1_EL1 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

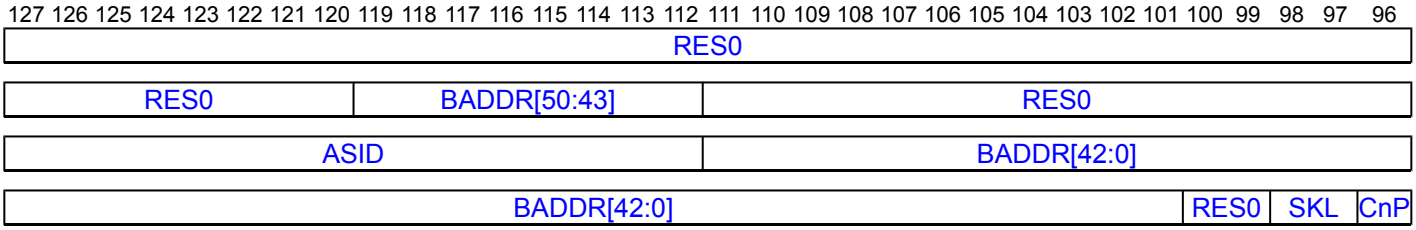
Attributes

TTBR1_EL1 is a:

- 128-bit register when FEAT_D128 is implemented and TCR2_EL1.D128 == 1
- 64-bit register when FEAT_D128 is not implemented or TCR2_EL1.D128 == 0

Field descriptions

When FEAT_D128 is implemented and TCR2_EL1.D128 == 1:



Bits [127:88]

Reserved, RES0.

BADDR, bits [87:80, 47:5]

Translation table base address:

- Bits A[55:x] of the stage 1 translation table base address bits are in register bits[87:80, 47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The BADDR field is split as follows:

- BADDR[50:43] is TTBR1_EL1[87:80].
- BADDR[42:0] is TTBR1_EL1[47:5].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [79:64]

Reserved, RES0.

ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level associated with translation table walks using TTBR1_EL1.

This determines the number of levels to be skipped from the regular start level of the stage 1 EL1&0 translation table walks using TTBR1_EL1.

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR1_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR1_EL1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.
0b1	The translation table entries pointed to by TTBR1_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR1_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

This bit is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR1_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

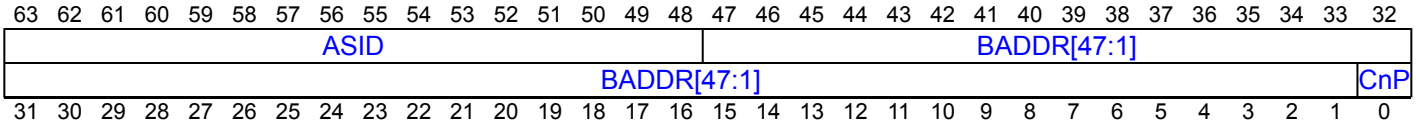
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When FEAT_D128 is not implemented or TCR2_EL1.D128 == 0:



ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL1.T1SZ, the translation stage, and the translation granule size.

If the value of TCR_EL1.IPS is not `0b110`, then:

- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits A[51:48] of the stage 1 translation table base address are `0b0000`.

The BADDR field represents a 52-bit address if one of the following applies:

- FEAT_LPA is implemented, the 64KB granule size is in use, and the value of TCR_EL1.IPS is `0b110`.
- FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of TCR_EL1.DS is 1.
- FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of TCR2_EL1.D128 is 0.

When TTBR1_EL1.BADDR represents a 52-bit addresses, all of the following apply:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- The smallest permitted value of x is 6.
- When x>6, register bits[(x-1):6] are RES0.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

For the 64KB granule, if FEAT_LPA is not implemented, and the value of TCR_EL1.IPS is `0b110`, one the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR1_EL1, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]
When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1.

CnP	Meaning
`0b0`	The translation table entries pointed to by TTBR1_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR1_EL1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.
`0b1`	The translation table entries pointed to by TTBR1_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR1_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

This bit is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR1_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR1_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TTBR1_EL1 or TTBR1_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TTBR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x210];
    else
        X[t, 64] = TTBR1_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR1_EL2<63:0>;
    else
        X[t, 64] = TTBR1_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR1_EL1<63:0>;
```

```
MSR TTBR1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x210] = X[t, 64];
    else
        TTBR1_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR1_EL2<63:0> = X[t, 64];
    else
        TTBR1_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR1_EL1<63:0> = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, TTBR1_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x210];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR1_EL1<63:0>;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR1_EL1<63:0>;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR TTBR1_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x210] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR1_EL1<63:0> = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TTBR1_EL1<63:0> = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_D128 is implemented

```
MRRS <Xt>, <Xt+1>, TTBR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, t2, 128] = NVMem[0x210, 128];
    else
        X[t, t2, 128] = TTBR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        X[t, t2, 128] = TTBR1_EL2;
    else
        X[t, t2, 128] = TTBR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = TTBR1_EL1;
```

When FEAT_D128 is implemented

```
MSRR TTBR1_EL1, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x210, 128] = X[t, t2, 128];
    else
        TTBR1_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        TTBR1_EL2<127:0> = X[t, t2, 128];
    else
        TTBR1_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR1_EL1<127:0> = X[t, t2, 128];
```

When FEAT_D128 is implemented and FEAT_VHE is implemented

```
MRRS <Xt>, <Xt+1>, TTBR1_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, t2, 128] = NVMem[0x210, 128];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            X[t, t2, 128] = TTBR1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, t2, 128] = TTBR1_EL1;
    else
        UNDEFINED;
```

When FEAT_D128 is implemented and FEAT_VHE is implemented

```
MSRR TTBR1_EL12, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x210, 128] = X[t, t2, 128];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            TTBR1_EL1<127:0> = X[t, t2, 128];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TTBR1_EL1<127:0> = X[t, t2, 128];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR1_EL2, Translation Table Base Register 1 (EL2)

The TTBR1_EL2 characteristics are:

Purpose

When the Effective value of HCR_EL2.E2H is 1, holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the EL2&0 translation regime, and other information for this translation regime.

Note

When the Effective value of HCR_EL2.E2H is not 1, the contents of this register are ignored by the PE, except for a direct read or write of the register.

Configuration

This register is present only when FEAT_VHE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TTBR1_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

TTBR1_EL2 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

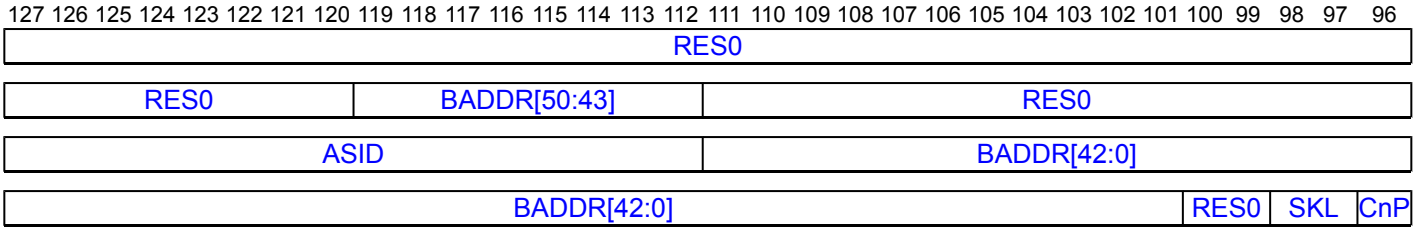
Attributes

TTBR1_EL2 is a:

- 128-bit register when FEAT_D128 is implemented, TCR2_EL2.D128 == 1, and ELIsInHost(EL2)
- 64-bit register when FEAT_D128 is not implemented or TCR2_EL2.D128 == 0

Field descriptions

When FEAT_D128 is implemented, TCR2_EL2.D128 == 1, and ELIsInHost(EL2):



Bits [127:88]

Reserved, RES0.

BADDR, bits [87:80, 47:5]

Translation table base address:

- Bits A[55:x] of the stage 1 translation table base address bits are in register bits[87:80, 47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The BADDR field is split as follows:

- BADDR[50:43] is TTBR1_EL2[87:80].
- BADDR[42:0] is TTBR1_EL2[47:5].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [79:64]

Reserved, RES0.

ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL2.A1 field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level associated with translation table walks using TTBR1_EL2.

This determines the number of levels to be skipped from the regular start level of the stage 1 EL2&0 translation table walks using TTBR1_EL2.

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR1_EL2 for the current ASID are permitted to differ from corresponding entries for TTBR1_EL2 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1_EL2.CnP on those other PEs. • The value of the current ASID.
0b1	The translation table entries pointed to by TTBR1_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR1_EL2. • The ASID is the same as the current ASID.

This bit is permitted to be cached in a TLB.

Note

- TTBR1_EL2 is accessible only when the Effective value of HCR_EL2.E2H is 1, meaning the current translation regime is the EL2&0 regime.
- If the value of the TTBR1_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

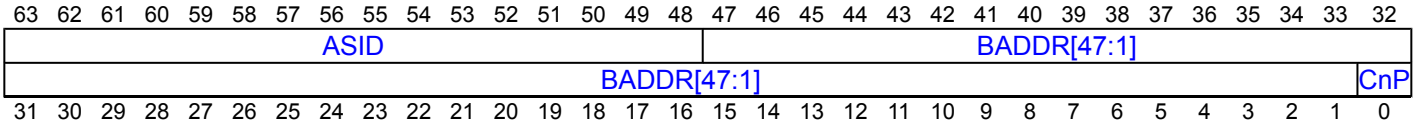
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When FEAT_D128 is not implemented or TCR2_EL2.D128 == 0:



ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL2.A1 field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL2.T1SZ, the translation stage, and the translation granule size.

If the value of TCR_EL2.{I}PS is not `0b110`, then:

- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits A[51:48] of the stage 1 translation table base address are `0b0000`.

The BADDR field represents a 52-bit address if one of the following applies:

- FEAT_LPA is implemented, the 64KB granule size is in use, and the value of TCR_EL2.{I}PS is `0b110`.
- FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of TCR_EL2.DS is 1.
- FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of TCR2_EL2.D128 is 0.

When TTBR1_EL2.BADDR represents a 52-bit addresses, all of the following apply:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- The smallest permitted value of x is 6.
- When x>6, register bits[(x-1):6] are RES0.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

The OA size specified by TCR_EL2.{I}PS is determined as follows:

- The value of TCR_EL2.PS when the Effective value of HCR_EL2.E2H is not 1.
- The value of TCR_EL2.IPS when the Effective value of HCR_EL2.E2H is 1.

For the 64KB granule, if FEAT_LPA is not implemented, and the value of TCR_EL2.{I}PS is `0b110`, one the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR1_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1.

CnP	Meaning
`0b0`	The translation table entries pointed to by TTBR1_EL2 for the current ASID are permitted to differ from corresponding entries for TTBR1_EL2 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1_EL2.CnP on those other PEs. • The value of the current ASID.
`0b1`	The translation table entries pointed to by TTBR1_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR1_EL2. • The ASID is the same as the current ASID.

This bit is permitted to be cached in a TLB.

Note

- TTBR1_EL2 is accessible only when the Effective value of HCR_EL2.E2H is 1, meaning the current translation regime is the EL2&0 regime.
- If the value of the TTBR1_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR1_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TTBR1_EL2 or TTBR1_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TTBR1_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TTBR1_EL2<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR1_EL2<63:0>;
```

```
MSR TTBR1_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TTBR1_EL2<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR1_EL2<63:0> = X[t, 64];
```

```
MRS <Xt>, TTBR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x210];
    else
        X[t, 64] = TTBR1_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR1_EL2<63:0>;
    else
        X[t, 64] = TTBR1_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR1_EL1<63:0>;
```

```
MSR TTBR1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x210] = X[t, 64];
    else
        TTBR1_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR1_EL2<63:0> = X[t, 64];
    else
        TTBR1_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR1_EL1<63:0> = X[t, 64];
```

When FEAT_D128 is implemented

```
MRRS <Xt>, <Xt+1>, TTBR1_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = TTBR1_EL2;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = TTBR1_EL2;
```

When FEAT_D128 is implemented

```
MSRR TTBR1_EL2, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        TTBR1_EL2<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR1_EL2<127:0> = X[t, t2, 128];
```

When FEAT_D128 is implemented

```
MRRS <Xt>, <Xt+1>, TTBR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, t2, 128] = NVMem[0x210, 128];
    else
        X[t, t2, 128] = TTBR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        X[t, t2, 128] = TTBR1_EL2;
    else
        X[t, t2, 128] = TTBR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = TTBR1_EL1;
```

When FEAT_D128 is implemented

```
MSRR TTBR1_EL1, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x210, 128] = X[t, t2, 128];
    else
        TTBR1_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        TTBR1_EL2<127:0> = X[t, t2, 128];
    else
        TTBR1_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR1_EL1<127:0> = X[t, t2, 128];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

UAO, User Access Override

The UAO characteristics are:

Purpose

Allows access to the User Access Override bit.

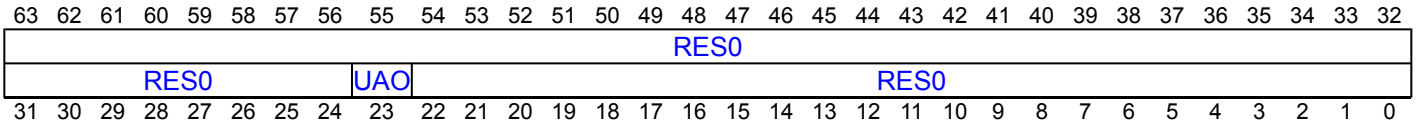
Configuration

This register is present only when FEAT_UAO is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to UAO are UNDEFINED.

Attributes

UAO is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

UAO, bit [23]

User Access Override.

UAO	Meaning
0b0	The behavior of LDTR* and STTR* instructions is as defined in the base Armv8 architecture.
0b1	When executed at the following Exception levels, LDTR* and STTR* instructions behave as the equivalent LDR* and STR* instructions: • EL1. • EL2 when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

When executed at EL3, or at EL2 when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the LDTR* and STTR* instructions behave as the equivalent LDR* and STR* instructions, regardless of the setting of the PSTATE.UAO bit.

Bits [22:0]

Reserved, RES0.

Accessing UAO

For more information about the operation of the MSR (immediate) accessor, see 'MSR (immediate)'.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, UAO

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_UAO) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    X[t, 64] = Zeros(40):PSTATE.UAO:Zeros(23);
elsif PSTATE.EL == EL2 then
    X[t, 64] = Zeros(40):PSTATE.UAO:Zeros(23);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Zeros(40):PSTATE.UAO:Zeros(23);
```

MSR UAO, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0100	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_UAO) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    PSTATE.UAO = X[t, 64]<23>;
elsif PSTATE.EL == EL2 then
    PSTATE.UAO = X[t, 64]<23>;
elsif PSTATE.EL == EL3 then
    PSTATE.UAO = X[t, 64]<23>;
```

MSR UAO, #<imm>

op0	op1	CRn	op2
0b00	0b000	0b0100	0b011

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VBAR_EL1, Vector Base Address Register (EL1)

The VBAR_EL1 characteristics are:

Purpose

Holds the vector base address for any exception that is taken to EL1.

Configuration

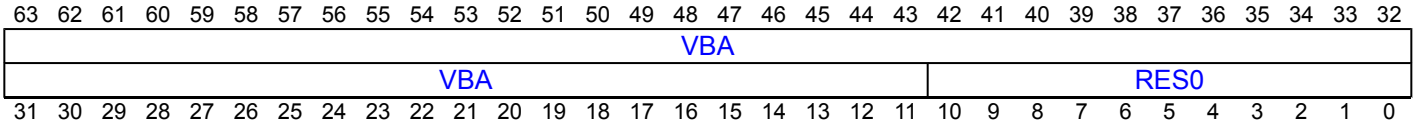
AArch64 System register VBAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register VBAR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VBAR_EL1 are UNDEFINED.

Attributes

VBAR_EL1 is a 64-bit register.

Field descriptions



VBA, bits [63:11]

Vector Base Address. Base address of the exception vectors for exceptions taken to EL1.

Note

If the implementation supports FEAT_LVA3, then:

- If tagged addresses are not being used, bits [63:56] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

Otherwise:

If the implementation supports FEAT_LVA, then:

- If tagged addresses are being used, bits [55:52] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:52] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

If the implementation does not support FEAT_LVA, then:

- If tagged addresses are being used, bits [55:48] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:48] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:0]

Reserved, RES0.

Accessing VBAR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name VBAR_EL1 or VBAR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VBAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x250];
    else
        X[t, 64] = VBAR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = VBAR_EL2;
    else
        X[t, 64] = VBAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VBAR_EL1;
```

```
MSR VBAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x250] = X[t, 64];
    else
        VBAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        VBAR_EL2 = X[t, 64];
    else
        VBAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VBAR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, VBAR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x250];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = VBAR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = VBAR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR VBAR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x250] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        VBAR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        VBAR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VBAR_EL2, Vector Base Address Register (EL2)

The VBAR_EL2 characteristics are:

Purpose

Holds the vector base address for any exception that is taken to EL2.

Configuration

AArch64 System register VBAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HVBAR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VBAR_EL2 are UNDEFINED.

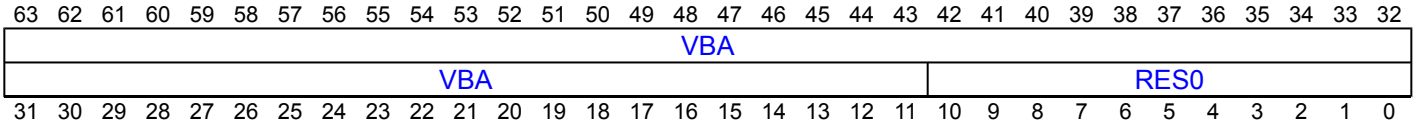
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VBAR_EL2 is a 64-bit register.

Field descriptions



VBA, bits [63:11]

Vector Base Address. Base address of the exception vectors for exceptions taken to EL2.

Note

If FEAT_LVA3 is implemented:

- If the Effective value of HCR_EL2.E2H is 1:
  - If tagged addresses are not being used, bits [63:56] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
- If the Effective value of HCR_EL2.E2H is not 1:
  - If tagged addresses are not being used, bits [63:56] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.

Otherwise :

If FEAT_LVA is implemented:

- If the Effective value of HCR_EL2.E2H is 1:
  - If tagged addresses are being used, bits [55:52] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
  - If tagged addresses are not being used, bits [63:52] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
- If the Effective value of HCR_EL2.E2H is not 1:
  - If tagged addresses are being used, bits [55:52] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.
  - If tagged addresses are not being used, bits [63:52] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.

If FEAT_LVA is not implemented:

- If the Effective value of HCR_EL2.E2H is 1:
  - If tagged addresses are being used, bits [55:48] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
  - If tagged addresses are not being used, bits [63:48] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
- If the Effective value of HCR_EL2.E2H is not 1:
  - If tagged addresses are being used, bits [55:48] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.
  - If tagged addresses are not being used, bits [63:48] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:0]

Reserved, RES0.

Accessing VBAR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name VBAR_EL2 or VBAR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VBAR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VBAR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VBAR_EL2;
```

```
MSR VBAR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VBAR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VBAR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, VBAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGRTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x250];
    else
        X[t, 64] = VBAR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = VBAR_EL2;
    else
        X[t, 64] = VBAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VBAR_EL1;
```

When FEAT_VHE is implemented

```
MSR VBAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
&& HFGWTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x250] = X[t, 64];
    else
        VBAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        VBAR_EL2 = X[t, 64];
    else
        VBAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VBAR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VBAR_EL3, Vector Base Address Register (EL3)

The VBAR_EL3 characteristics are:

Purpose

Holds the vector base address for any exception that is taken to EL3.

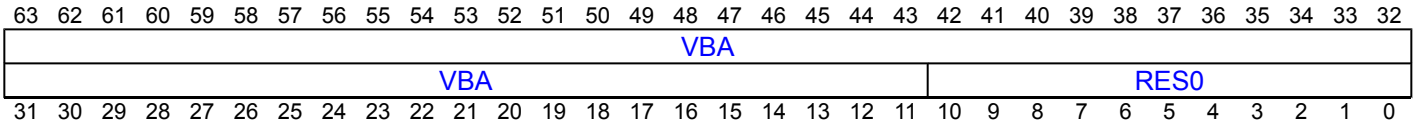
Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VBAR_EL3 are UNDEFINED.

Attributes

VBAR_EL3 is a 64-bit register.

Field descriptions



VBA, bits [63:11]

Vector Base Address. Base address of the exception vectors for exceptions taken to EL3.

Note

If the implementation supports FEAT_LVA3, then:

- If tagged addresses are not being used, bits [63:56] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.

Otherwise:

If the implementation supports FEAT_LVA, then:

- If tagged addresses are being used, bits [55:52] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:52] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.

If the implementation does not support FEAT_LVA, then:

- If tagged addresses are being used, bits [55:48] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:48] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:0]

Reserved, RES0.

Accessing VBAR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VBAR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VBAR_EL3;
```

```
MSR VBAR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.VBAR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        VBAR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VDISR_EL2, Virtual Deferred Interrupt Status Register (EL2)

The VDISR_EL2 characteristics are:

Purpose

Records that a virtual SError exception has been consumed by an ESB instruction executed at EL1.

An indirect write to VDISR_EL2 made by an ESB instruction does not require an explicit synchronization operation for the value written to be observed by a direct read of one of the following registers occurring in program order after the ESB instruction:

- DISR_EL1.
- DISR.

Configuration

AArch64 System register VDISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VDISR[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to VDISR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

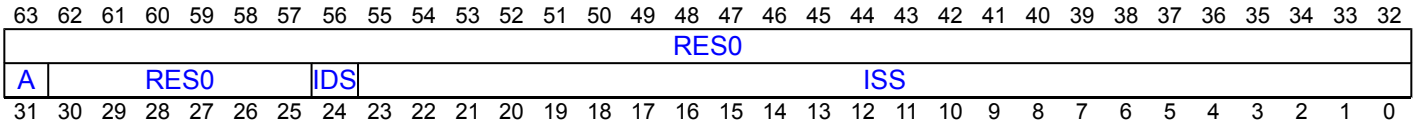
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VDISR_EL2 is a 64-bit register.

Field descriptions

When EL1 is using AArch64:



Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

The value copied from VSESR_EL2.IDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

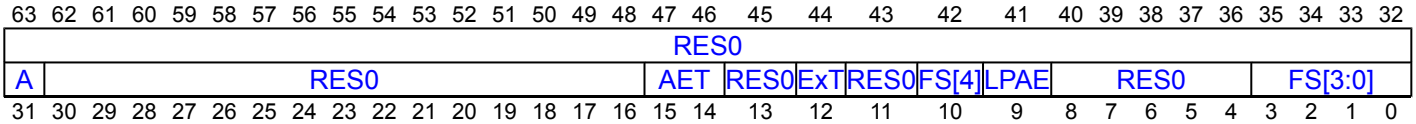
ISS, bits [23:0]

The value copied from VSESR_EL2.ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When EL1 is using AArch32 and VDISR_EL2.LPAE == 0:



Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from VSESR_EL2.AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from VSESR_EL2.ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault status code. Set to `0b10110` when an `ESB` instruction defers a virtual SError exception.

FS	Meaning
`0b10110`	Asynchronous SError exception.

All other values are reserved.

The FS field is split as follows:

- FS[4] is VDISR_EL2[10].
- FS[3:0] is VDISR_EL2[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

Format.

Set to TTBCR.EAE when an `ESB` instruction defers a virtual SError exception.

LPAE	Meaning
`0b0`	Using the Short-descriptor translation table format.

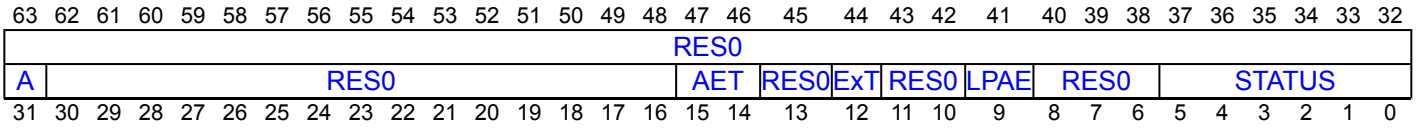
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When EL1 is using AArch32 and VDISR_EL2.LPAE == 1:



Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an `ESB` instruction defers a virtual SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from VSESR_EL2.AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from VSESR_EL2.ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

Format.

Set to TTBCR.EAE when an ESB instruction defers a virtual SError exception.

LPAE	Meaning
0b1	Using the Long-descriptor translation table format.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status code. Set to 0b010001 when an ESB instruction defers a virtual SError exception.

STATUS	Meaning
0b010001	Asynchronous SError exception.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VDISR_EL2

An indirect write to VDISR_EL2 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of one of the following registers occurring in program order after the ESB instruction:

- DISR_EL1.
- DISR.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VDISR_EL2
```

op0	op1	CRn	CRm	op2

0b11	0b100	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x500];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VDISR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VDISR_EL2;
```

MSR VDISR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x500] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VDISR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VDISR_EL2 = X[t, 64];
```

MRS <Xt>, DISR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        X[t, 64] = VDISR_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DISR_EL1;
```

MSR DISR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        VDISR_EL2 = X[t, 64];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VDISR_EL3, Virtual Deferred Interrupt Status Register (EL3)

The VDISR_EL3 characteristics are:

Purpose

Records that a delegated SError exception has been consumed by an ESB instruction executed at EL2 or EL1 when the Effective value of SCR_EL3.DSE is 1.

An indirect write to VDISR_EL3 made by an ESB instruction does not require an explicit synchronization operation for the value written to be observed by a direct read of DISR_EL1 occurring in program order after the ESB instruction.

Configuration

This register is present only when FEAT_E3DSE is implemented. Otherwise, direct accesses to VDISR_EL3 are UNDEFINED.

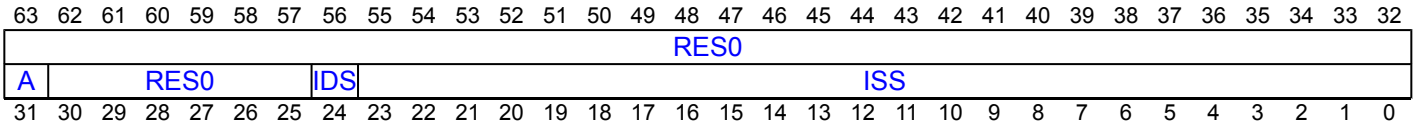
Note

The encoding for this register is subject to change.

Attributes

VDISR_EL3 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a delegated SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

The value copied from VSESR_EL3.IDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

The value copied from VSESR_EL3.ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VDISR_EL3

An indirect write to VDISR_EL3 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR_EL1 occurring in program order after the ESB instruction.

Note

The accessibility pseudocode for DISR_EL1 has not been updated to show the effect of VDISR_EL3.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VDISR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_E3DSE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VDISR_EL3;
```

```
MSR VDISR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_E3DSE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    VDISR_EL3 = X[t, 64];
```

```
MRS <Xt>, DISR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        X[t, 64] = VDISR_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DISR_EL1;
```

MSR DISR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        VDISR_EL2 = X[t, 64];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VMECID_A_EL2, Alternate MECID for EL1&0 stage 2 translation regime

The VMECID_A_EL2 characteristics are:

Purpose

Alternate MECID for EL1&0 stage 2 translation regime.

Configuration

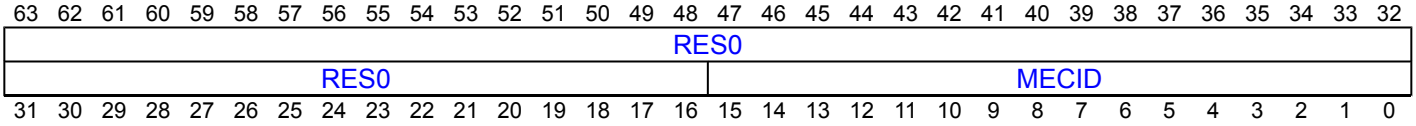
This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VMECID_A_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

VMECID_A_EL2 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VMECID_A_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VMECID_A_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1001	0b001

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = VMECID_A_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VMECID_A_EL2;
```

MSR VMECID_A_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1001	0b001

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        VMECID_A_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VMECID_A_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VMECID_P_EL2, Primary MECID for EL1&0 stage 2 translation regime

The VMECID_P_EL2 characteristics are:

Purpose

Primary MECID for EL1&0 stage 2 translation regime.

Configuration

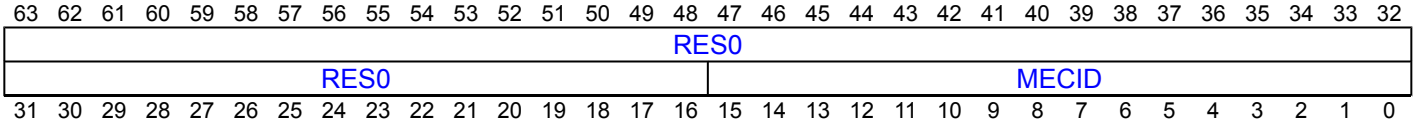
This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VMECID_P_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

VMECID_P_EL2 is a 64-bit register.

Field descriptions



Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VMECID_P_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VMECID_P_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1001	0b000

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = VMECID_P_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VMECID_P_EL2;
```

```
MSR VMECID_P_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1001	0b000

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        VMECID_P_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VMECID_P_EL2 = X[t, 64];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VMPIDR_EL2, Virtualization Multiprocessor ID Register

The VMPIDR_EL2 characteristics are:

Purpose

Holds the value of the Virtualization Multiprocessor ID. This is the value returned by EL1 reads of MPIDR_EL1, which in a multiprocessor system, provides an additional PE identification system.

Configuration

AArch64 System register VMPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VMPIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VMPIDR_EL2 are UNDEFINED.

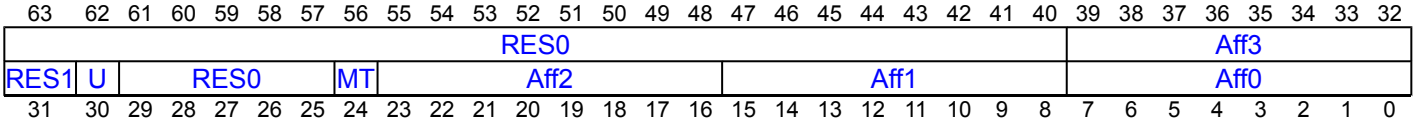
If EL2 is not implemented, reads of this register return the value of the MPIDR_EL1 and writes to the register are ignored.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VMPIDR_EL2 is a 64-bit register.

Field descriptions



Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3. See the description of VMPIDR_EL2.Aff0 for more information.

Aff3 is not supported in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [31]

Reserved, RES1.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of VMPIDR_EL2.Aff0 for more information about affinity levels.

MT	Meaning
0b0	Performance of PEs at the lowest affinity level is largely independent.
0b1	Performance of PEs at the lowest affinity level is very interdependent.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Aff2, bits [23:16]

Affinity level 2. See the description of VMPIDR_EL2.Aff0 for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Aff1, bits [15:8]

Affinity level 1. See the description of VMPIDR_EL2.Aff0 for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Aff0, bits [7:0]

Affinity level 0.

The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VMPIDR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VMPIDR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x050];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VMPIDR_EL2;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        X[t, 64] = MPIDR_EL1;
    else
        X[t, 64] = VMPIDR_EL2;
```

MSR VMPIDR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x050] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VMPIDR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return;
    else
        VMPIDR_EL2 = X[t, 64];
```

MRS <Xt>, MPIDR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.MPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() then
        X[t, 64] = VMPIDR_EL2;
    else
        X[t, 64] = MPIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MPIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VNCR_EL2, Virtual Nested Control Register

The VNCR_EL2 characteristics are:

Purpose

When FEAT_NV2 is implemented, holds the base address that is used to define the memory location that is accessed by transformed reads and writes of System registers.

Configuration

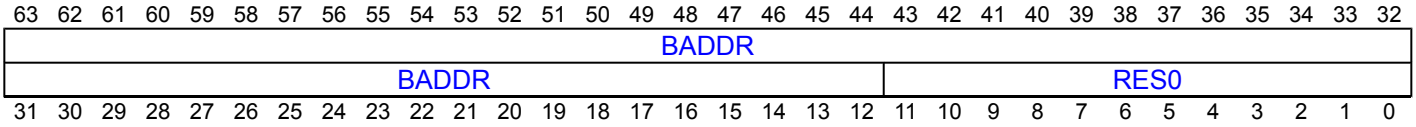
This register is present only when FEAT_NV2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VNCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VNCR_EL2 is a 64-bit register.

Field descriptions



BADDR, bits [63:12]

Base Address. When a register read/write is transformed to be a Load or Store, the address of the load/store is to VNCR_EL2.BADDR:Offset<11:0>.

Bits[63:n] are RESS where n is one of the following:

- If FEAT_LVA3 is implemented, n is 57.
- Otherwise, if FEAT_LVA is implemented, n is 53.
- If FEAT_LVA is not implemented, n is 49.

If the bits marked as RESS do not all have the same value as bit[n-1], then there is a CONSTRAINED UNPREDICTABLE choice between:

- Generating a Translation fault when translating the transformed register read/write.
- The bits behave as the same value as bit[n-1] for all purposes other than reading back the register.
- The bits are treated as the same value as bit[n-1] for all purposes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

Accessing VNCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VNCR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0010	0b000

```
if !(IsFeatureImplemented(FEAT_NV2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x0B0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VNCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VNCR_EL2;
```

MSR VNCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0010	0b000

```
if !(IsFeatureImplemented(FEAT_NV2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x0B0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VNCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VNCR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VPIDR_EL2, Virtualization Processor ID Register

The VPIDR_EL2 characteristics are:

Purpose

Holds the value of the Virtualization Processor ID. This is the value returned by EL1 reads of MIDR_EL1.

Configuration

AArch64 System register VPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VPIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VPIDR_EL2 are UNDEFINED.

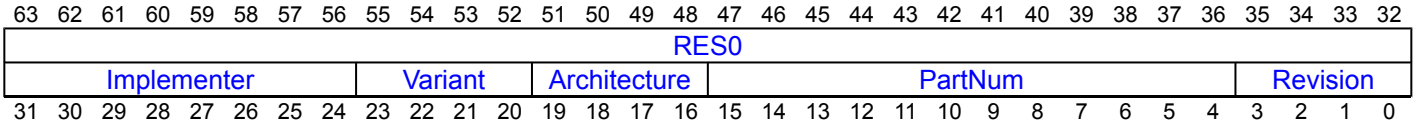
If EL2 is not implemented, reads of this register return the value of the MIDR_EL1 and writes to the register are ignored.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VPIDR_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm.

Implementer	Meaning
0x00	Reserved for software use.
0x41	Arm Limited.
0x42	Broadcom Corporation.
0x43	Cavium Inc.
0x44	Digital Equipment Corporation.
0x46	Fujitsu Ltd.
0x49	Infineon Technologies AG.
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation.
0x50	Applied Micro Circuits Corporation.
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation.
0xC0	Ampere Computing.

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Variant, bits [23:20]

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Architecture, bits [19:16]

Architecture version.

Architecture	Meaning
0b0001	Armv4.
0b0010	Armv4T.
0b0011	Armv5 (obsolete).
0b0100	Armv5T.
0b0101	Armv5TE.
0b0110	Armv5TEJ.
0b0111	Armv6.
0b1111	Architectural features are individually identified in the ID_* registers.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PartNum, bits [15:4]

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Revision, bits [3:0]

An IMPLEMENTATION DEFINED revision number for the device.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VPIDR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VPIDR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x088];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VPIDR_EL2;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        X[t, 64] = MIDR_EL1;
    else
        X[t, 64] = VPIDR_EL2;
```

MSR VPIDR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x088] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VPIDR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return;
    else
        VPIDR_EL2 = X[t, 64];
```

MRS <Xt>, MIDR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.MIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() then
        X[t, 64] = VPIDR_EL2;
    else
        X[t, 64] = MIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MIDR_EL1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VSESR_EL2, Virtual SError Exception Syndrome Register (EL2)

The VSESR_EL2 characteristics are:

Purpose

Provides the syndrome value reported to software on taking a virtual SError exception to EL1, or on executing an ESB instruction at EL1.

When the virtual SError exception injected using HCR_EL2.VSE is taken to EL1 using AArch64, then the syndrome value is reported in ESR_EL1.

When the virtual SError exception injected using HCR_EL2.VSE is taken to EL1 using AArch32, then the syndrome value is reported in DFSR.{AET, ExT} and the remainder of DFSR is set as defined by VMSAv8-32. For more information, see The AArch32 Virtual Memory System Architecture.

When the virtual SError exception injected using HCR_EL2.VSE is deferred by an ESB instruction, then the syndrome value is written to VDISR_EL2.

Configuration

AArch64 System register VSESR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VDFSR[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to VSESR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

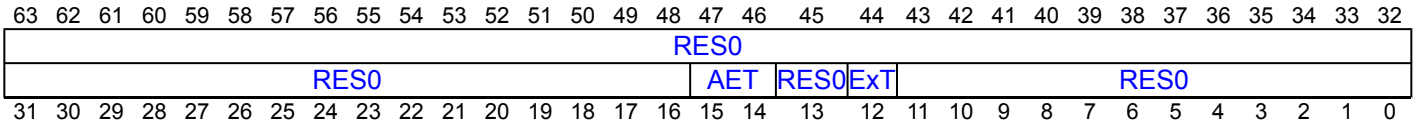
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VSESR_EL2 is a 64-bit register.

Field descriptions

When EL1 is using AArch32:



Bits [63:16]

Reserved, RES0.

AET, bits [15:14]

When a virtual SError exception is taken to EL1 using AArch32, DFSR[15:14] is set to VSESR_EL2.AET.

When a virtual SError exception is deferred by an ESB instruction, VDISR_EL2[15:14] is set to VSESR_EL2.AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

When a virtual SError exception is taken to EL1 using AArch32, DFSR[12] is set to VSESR_EL2.ExT.

When a virtual SError exception is deferred by an `ESB` instruction, VDISR_EL2[12] is set to VSESR_EL2.ExT.

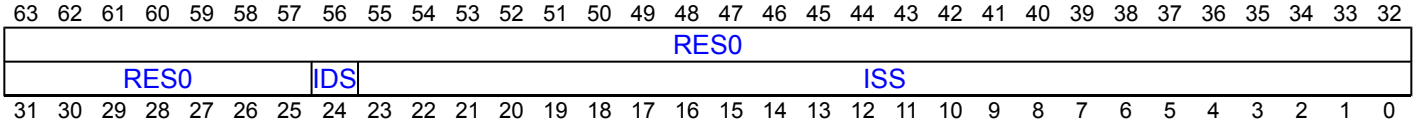
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

When EL1 is using AArch64:



Bits [63:25]

Reserved, RES0.

IDS, bit [24]

When a virtual SError exception is taken to EL1 using AArch64, ESR_EL1[24] is set to VSESR_EL2.IDS.

When a virtual SError exception is deferred by an `ESB` instruction, VDISR_EL2[24] is set to VSESR_EL2.IDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

When a virtual SError exception is taken to EL1 using AArch64, ESR_EL1[23:0] is set to VSESR_EL2.ISS.

When a virtual SError exception is deferred by an `ESB` instruction, VDISR_EL2[23:0] is set to VSESR_EL2.ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VSESR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VSESR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x508];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VSESR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VSESR_EL2;
```

MSR VSESR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x508] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VSESR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VSESR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VSESR_EL3, Virtual SError Exception Syndrome Register (EL3)

The VSESR_EL3 characteristics are:

Purpose

Provides the syndrome value reported to software when the Effective value of SCR_EL3.DSE is 1 on taking a delegated SError exception to EL2 or EL1, or on executing an `ESB` instruction at EL2 or EL1.

When the delegated SError exception injected using SCR_EL3.DSE is taken to EL2 using AArch64, then the syndrome value is reported in ESR_EL2.

When the delegated SError exception injected using SCR_EL3.DSE is taken to EL1 using AArch64, then the syndrome value is reported in ESR_EL1.

When the delegated SError exception injected using SCR_EL3.DSE is deferred by an `ESB` instruction, then the syndrome value is written to VDISR_EL3.

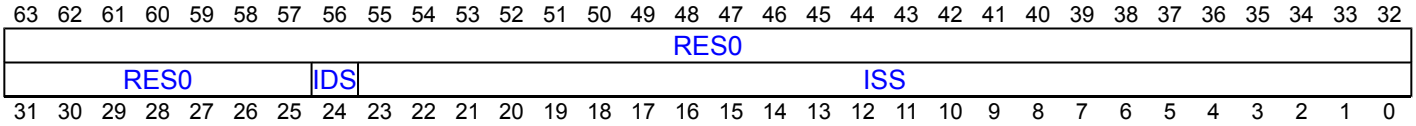
Configuration

This register is present only when FEAT_E3DSE is implemented. Otherwise, direct accesses to VSESR_EL3 are UNDEFINED.

Attributes

VSESR_EL3 is a 64-bit register.

Field descriptions



Bits [63:25]

Reserved, RES0.

IDS, bit [24]

When a delegated SError exception triggered by SCR_EL3.DSE is taken to EL2 or EL1 using AArch64, ESR_ELx[24] is set to VSESR_EL3.IDS.

When a delegated SError exception triggered by SCR_EL3.DSE is deferred by an `ESB` instruction, VDISR_EL3[24] is set to VSESR_EL3.IDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

When a delegated SError exception triggered by SCR_EL3.DSE is taken to EL2 or EL1 using AArch64, ESR_ELx[23:0] is set to VSESR_EL3.ISS.

When a delegated SError exception triggered by SCR_EL3.DSE is deferred by an `ESB` instruction, VDISR_EL3[23:0] is set to VSESR_EL3.ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VSESR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VSESR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_E3DSE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VSESR_EL3;
```

```
MSR VSESR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_E3DSE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    VSESR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VSTCR_EL2, Virtualization Secure Translation Control Register

The VSTCR_EL2 characteristics are:

Purpose

The control register for stage 2 of the Secure EL1&0 translation regime.

Configuration

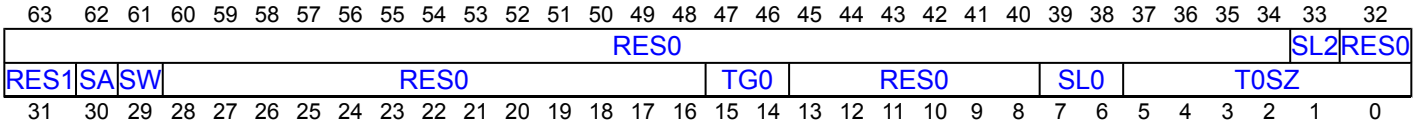
This register is present only when FEAT_SEL2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VSTCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VSTCR_EL2 is a 64-bit register.

Field descriptions



Any of the bits in VSTCR_EL2 are permitted to be cached in a TLB.

Bits [63:34]

Reserved, RES0.

SL2, bit [33]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == 0):

Starting level of the Secure stage 2 translation lookup controlled by VSTCR_EL2.

If VTCR_EL2.DS == 1, then VSTCR_EL2.SL2, in combination with VSTCR_EL2.SL0, gives encodings for the Secure stage 2 translation table walk initial lookup level.

If VTCR_EL2.DS == 0, then VSTCR_EL2.SL2 is RES0.

If the translation granule size is not 4KB, then this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [32]

Reserved, RES0.

Bit [31]

Reserved, RES1.

SA, bit [30]

Secure stage 2 translation output address space.

SA	Meaning
0b0	All stage 2 translations for the Secure IPA space access the Secure PA space.
0b1	All stage 2 translations for the Secure IPA space access the Non-secure PA space.

When the value of VSTCR_EL2.SW is 1, this bit behaves as 1 for all purposes other than reading back the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SW, bit [29]

Secure stage 2 translation address space.

SW	Meaning
0b0	All stage 2 translation table walks for the Secure IPA space are to the Secure PA space.
0b1	All stage 2 translation table walks for the Secure IPA space are to the Non-secure PA space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [28:16]

Reserved, RES0.

TG0, bits [15:14]

Secure stage 2 granule size for VSTTBR_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If FEAT_GTG is implemented, ID_AA64MMFR0_EL1.{TGran4_2, TGran16_2, TGran64_2} indicate which granule sizes are supported for stage 2 translation.

If FEAT_GTG is not implemented, ID_AA64MMFR0_EL1.{TGran4, TGran16, TGran64} indicate which granule sizes are supported.

If the value is programmed to either a reserved value, or a size that has not been implemented, then for all purposes other than read back from this register, the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [13:8]

Reserved, RES0.

SL0, bits [7:6]

When FEAT_TTST is implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == 0):

Starting level of the Secure stage 2 translation lookup, controlled by VSTCR_EL2. The meaning of this field depends on the value of VSTCR_EL2.TG0.

SL0	Meaning
0b00	If VSTCR_EL2.TG0 is 0b00 (4KB granule): • If FEAT_LPA2 is not implemented, start at level 2. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is 0b0, start at level 2. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is 0b1, start at level -1. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.
0b01	If VSTCR_EL2.TG0 is 0b00 (4KB granule): • If FEAT_LPA2 is not implemented, start at level 1. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is 0b0, start at level 1. • If FEAT_LPA2 is implemented, the combination of VSTCR_EL2.SL0 == 01 and VSTCR_EL2.SL2 == 1 is reserved. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.
0b10	If VSTCR_EL2.TG0 is 0b00 (4KB granule): • If FEAT_LPA2 is not implemented, start at level 0. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is 0b0, start at level 0. • If FEAT_LPA2 is implemented, the combination of VSTCR_EL2.SL0 == 10 and VSTCR_EL2.SL2 == 1 is reserved. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.
0b11	If VSTCR_EL2.TG0 is 0b00 (4KB granule): • If FEAT_LPA2 is not implemented, start at level 3. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is 0b0, start at level 3. • If FEAT_LPA2 is implemented, the combination of VSTCR_EL2.SL0 == 11 and VSTCR_EL2.SL2 == 1 is reserved. If VSTCR_EL2.TG0 is 0b10 (16KB granule) and FEAT_LPA2 is implemented, start at level 0.

If this field is programmed to a value that is not consistent with the programming of VSTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TTST is not implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == 0):

Starting level of the Secure stage 2 translation lookup, controlled by VSTCR_EL2. The meaning of this field depends on the value of VSTCR_EL2.TG0.

SL0	Meaning
0b00	If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.
0b01	If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.
0b10	If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VSTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by VSTTBR_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for this field depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

If this field is programmed to a value that is not consistent with the programming of SL0, then a stage 2 level 0 Translation fault is generated.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VSTCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VSTCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0110	0b010

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x048];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = VSTCR_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = VSTCR_EL2;
```

```
MSR VSTCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0110	0b010

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x048] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        VSTCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        VSTCR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VSTTBR_EL2, Virtualization Secure Translation Table Base Register

The VSTTBR_EL2 characteristics are:

Purpose

The base register for stage 2 translation tables to translate Secure IPAs in the Secure EL1&0 translation regime. Holds the base address of the translation table for the initial lookup for stage 2 of an address translation for a Secure IPA in the Secure EL1&0 translation regime, and other information for this translation stage.

Configuration

This register is present only when FEAT_SEL2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VSTTBR_EL2 are UNDEFINED.

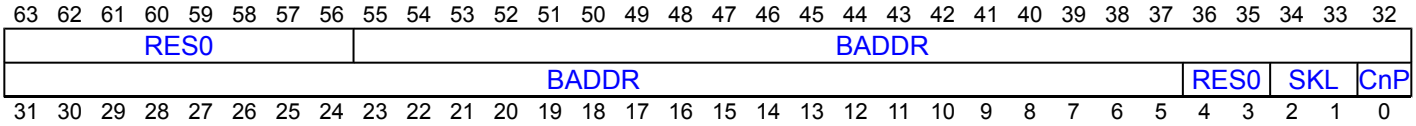
This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VSTTBR_EL2 is a 64-bit register.

Field descriptions

When FEAT_D128 is implemented and VTCR_EL2.D128 == 1:



Bits [63:56]

Reserved, RES0.

BADDR, bits [55:5]

- Bits A[55:x] of the stage 2 translation table base address bits are in register bits[55:x].
- Bits A[(x-1):0] of the stage 2 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level. Skip Level determines the number of levels to be skipped from the regular start level of the Secure stage 2 translation table walk.

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

Common not Private, for stage 2 of the Secure EL1&0 translation regime. In an implementation that includes FEAT_TTCNP, indicates whether each entry that is pointed to by VSTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by VSTTBR_EL2 are permitted to differ from the entries for VSTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
0b1	The translation table entries pointed to by VSTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.

This bit is permitted to be cached in a TLB.

Note

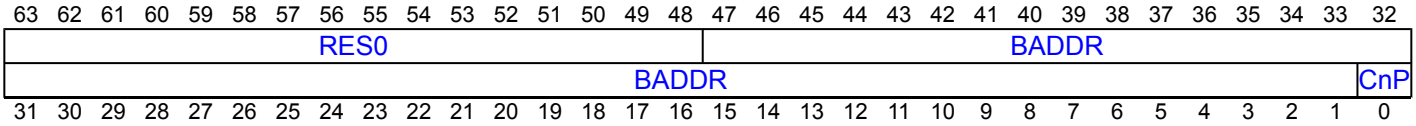
If the value of VSTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VSTTBR_EL2s do not point to the same translation table entries when using the current VMID, then the results of translations using VSTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is not implemented or VTCR_EL2.D128 == 0:



Bits [63:48]

Reserved, RES0.

BADDR, bits [47:1]

The BADDR field represents a 52-bit address if one of the following applies:

- FEAT_LPA is implemented, the 64KB granule size is in use, and the value of VTCR_EL2.PS is 0b110.
- FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of VTCR_EL2.DS is 1.
- FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of VTCR_EL2.D128 is 0.

When VSTTBR_EL2.BADDR represents a 52-bit addresses, all of the following apply:

- Register bits[47:x] hold bits[47:x] of the stage 2 translation table base address, where x is determined by the size of the translation table at the start level.

- The smallest permitted value of x is 6.
- Register bits[5:2] hold bits[51:48] of the stage 2 translation table base address.
- Bits[x:0] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

For the 64KB granule, if FEAT_LPA is not implemented, and the value of VTCR_EL2.PS is `0b110`, one the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits [55:52] of the stage 2 translation table base address are zero.

If the Effective value of VTCR_EL2.PS is not `0b110`, then:

- Register bits[47:x] hold bits[47:x] of the stage 2 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs then bits[51:48] of the translation table base addresses used in this stage of translation are `0b0000`.

If any VSTTBR_EL2[47:1] bit that is defined as RES0 has the value 1 when a translation table walk is performed using VSTTBR_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of VSTCR_EL2.T0SZ, the stage of translation, and the translation granule size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

Common not Private, for stage 2 of the Secure EL1&0 translation regime. In an implementation that includes FEAT_TTCNP, indicates whether each entry that is pointed to by VSTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1.

CnP	Meaning
`0b0`	The translation table entries pointed to by VSTTBR_EL2 are permitted to differ from the entries for VSTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
`0b1`	The translation table entries pointed to by VSTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.

This bit is permitted to be cached in a TLB.

Note

If the value of VSTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VSTTBR_EL2s do not point to the same translation table entries when using the current VMID, then the results of translations using VSTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VSTTBR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VSTTBR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x030];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = VSTTBR_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = VSTTBR_EL2;
```

```
MSR VSTTBR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x030] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        VSTTBR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        VSTTBR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VTCR_EL2, Virtualization Translation Control Register

The VTCR_EL2 characteristics are:

Purpose

The control register for stage 2 of the EL1&0 translation regime.

Configuration

AArch64 System register VTCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VTCR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VTCR_EL2 are UNDEFINED.

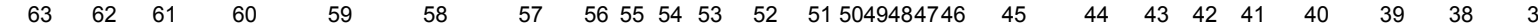
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VTCR_EL2 is a 64-bit register.

Field descriptions



Unless stated otherwise, any of the bits in VTCR_EL2 are permitted to be cached in a TLB.

Bits [63:46]

Reserved, RES0.

HDBSS, bit [45]

When FEAT_HDBSS is implemented:

Enable use of HDBSS.

HDBSS	Meaning
0b0	Hardware tracking of Dirty state Structure is disabled.
0b1	Hardware tracking of Dirty state Structure is enabled.

If VTCR_EL2.{HA, HD} is not {1, 1}, the Effective value of this field is 0.

If SCR_EL3.HDBSSEn is 0, then this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HAFT, bit [44]

When FEAT_HAFT is implemented:

Hardware managed Access Flag for Table descriptors.

Enables the Hardware managed Access Flag for Table descriptors.

HAFT	Meaning
0b0	Hardware managed Access Flag for Table descriptors is disabled.
0b1	Hardware managed Access Flag for Table descriptors is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [43:42]

Reserved, RES0.

TL0, bit [41]

When FEAT_THE is implemented:

Control bit to check for presence of MMU TopLevel0 permission attribute.

TL0	Meaning
0b0	This bit does not have any effect on stage 2 translations.
0b1	Enables MMU TopLevel0 permission attribute check for TTBR0_EL1 and TTBR1_EL1 translations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

GCSH, bit [40]

When FEAT_THE is implemented and FEAT_GCS is implemented:

Assured stage 1 translations for Guarded Control Stacks. Enforces use of the AssuredOnly attribute in stage 2 for the memory accessed by privileged Guarded Control Stack data accesses.

GCSH	Meaning
0b0	For the memory accessed by privileged Guarded Control Stack data accesses, the AssuredOnly attribute in stage 2 is not required to be set.
0b1	For the memory accessed by privileged Guarded Control Stack data accesses, the AssuredOnly attribute in stage 2 is required to be set.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [39]

Reserved, RES0.

D128, bit [38]

When FEAT_D128 is implemented:

Enables VMSAv9-128 translation system for stage 2 translation.

D128	Meaning
0b0	Translation system follows VMSAv8-64 translation process.
0b1	Translation system follows VMSAv9-128 translation process.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S2POE, bit [37]

When FEAT_S2POE is implemented:

Enable Permission Overlay. Enables permission overlay in stage 2 Permission model.

S2POE	Meaning
0b0	Overlay disabled.
0b1	Overaly enabled.

This bit is not permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S2PIE, bit [36]

When FEAT_S2PIE is implemented:

Select Permission Model. Enables usage of permission indirection in stage 2 Permission model.

S2PIE	Meaning
0b0	Direct permission model.
0b1	Indirect permission model.

This field is RES1 when VTCR_EL2.D128 is set.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TL1, bit [35]
When FEAT_THE is implemented:

Control bit to check for presence of MMU TopLevel1 permission attribute.

TL1	Meaning
0b0	This bit does not have any effect on stage 2 translations.
0b1	Enables MMU TopLevel1 permission attribute check for TTBR0_EL1 and TTBR1_EL1 translations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [34]
When FEAT_THE is implemented:

AssuredOnly attribute enable for VMSAv8-64. Configures use of bit[58] of the stage 2 translation table Block or Page descriptor.

AssuredOnly	Meaning
0b0	Bit[58] of each stage 2 translation Block or Page descriptor does not indicate AssuredOnly attribute.
0b1	Bit[58] of each stage 2 translation Block or Page descriptor indicates AssuredOnly attribute.

This field is RES0 when VTCR_EL2.D128 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SL2, bit [33]
When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == 0):

Starting level of the stage 2 translation lookup controlled by VTCR_EL2.

If VTCR_EL2.DS == 1, then VTCR_EL2.SL2, in combination with VTCR_EL2.SL0, gives encodings for the stage 2 translation table walk initial lookup level.

If VTCR_EL2.DS == 0, then VTCR_EL2.SL2 is RES0.

If the translation granule size is not 4KB, then this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [32]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == 0):

This field affects:

- Whether a 52-bit output address can be described by the translation tables of the 4KB or 16KB translation granules.
- The minimum value of VTCR_EL2.T0SZ and VSTCR_EL2.T0SZ.
- How and where shareability for Block and Page descriptors are encoded.

DS	Meaning
0b0	Bits[49:48] of translation descriptors are RES0. Bits[9:8] in Block and Page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in Table descriptors are ignored by hardware. The minimum value of VTCR_EL2.T0SZ is 16. Any memory access using a smaller value generates a stage 2 level 0 translation table fault. The minimum value of VSTCR_EL2.T0SZ is 16. Any memory access using a smaller value generates a stage 2 level 0 translation table fault. Output address[51:48] is 0000.
0b1	Bits[49:48] of translation descriptors hold output address[49:48]. Bits[9:8] in translation descriptors hold output address[51:50]. The shareability information of Block and Page descriptors for cacheable locations is determined by VTCR_EL2.SH0. The minimum value of VTCR_EL2.T0SZ is 12. Any memory access using a smaller value generates a stage 2 level 0 translation table fault. The minimum value of VSTCR_EL2.T0SZ is 12. Any memory access using a smaller value generates a stage 2 level 0 translation table fault. **Note** As FEAT_LPA must be implemented if VTCR_EL2.DS == 1, the minimum values of VTCR_EL2.T0SZ and VSTCR_EL2.T0SZ are 12, as determined by that extension. For the TLBI range instructions affecting IPA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 00. **Note** This forces alignment of the ranges used by the TLBI range instructions.

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES1.

NSA, bit [30]

When FEAT_SEL2 is implemented:

Non-secure stage 2 translation output address space for the Secure EL1&0 translation regime.

NSA	Meaning
0b0	All stage 2 translations for the Non-secure IPA space of the Secure EL1&0 translation regime access the Secure PA space.
0b1	All stage 2 translations for the Non-secure IPA space of the Secure EL1&0 translation regime access the Non-secure PA space.

This bit behaves as 1 for all purposes other than reading back the value of the bit when one of the following is true:

- The value of VTCR_EL2.NSW is 1.
- The value of VSTCR_EL2.SA is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSW, bit [29]

When FEAT_SEL2 is implemented:

Non-secure stage 2 translation table address space for the Secure EL1&0 translation regime.

NSW	Meaning
0b0	All stage 2 translation table walks for the Non-secure IPA space of the Secure EL1&0 translation regime are to the Secure PA space.
0b1	All stage 2 translation table walks for the Non-secure IPA space of the Secure EL1&0 translation regime are to the Non-secure PA space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 2 translation table Block or Page entry.

HWU62	Meaning
0b0	Bit[62] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[62] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 2 translation table Block or Page entry.

HWU61	Meaning
0b0	Bit[61] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[61] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 2 translation table Block or Page entry.

HWU60	Meaning
0b0	Bit[60] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 2 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[59] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [24:23]

Reserved, RES0.

HD, bit [22]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 2 translations when EL2 is enabled in the current Security state.

HD	Meaning
0b0	Stage 2 hardware management of dirty state disabled.
0b1	Stage 2 hardware management of dirty state enabled.

When the Effective value of VTCR_EL2.HA is 0, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 2 translations when EL2 is enabled in the current Security state.

HA	Meaning
0b0	Stage 2 Access flag update disabled.
0b1	Stage 2 Access flag update enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

VS, bit [19]

When FEAT_VMID16 is implemented:

VMID Size.

VS	Meaning
0b0	8-bit VMID. The upper 8 bits of VTTBR_EL2 are ignored by the hardware, and treated as if they are all zeros, for every purpose except when reading back the register.
0b1	16-bit VMID. The upper 8 bits of VTTBR_EL2 are used for allocation and matching in the TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PS, bits [18:16]

Physical address Size for the second stage of translation.

PS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.
0b111	56 bits, 64PB.

The value 0b110 represents the following output address sizes:

- For the 64KB translation granule size, if FEAT_LPA is implemented, the value 0b110 represents 52 bits.
- For the 4KB and 16KB translation granule sizes, if FEAT_LPA is implemented and the Effective value of VTCR_EL2.DS is 0b1, then the value 0b110 represents 52 bits.
- Otherwise, the value 0b110 behaves as the 0b101 value and represents 48 bits.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the VTTBR_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If FEAT_GTG is implemented, ID_AA64MMFR0_EL1.{TGran4_2, TGran16_2, TGran64_2} indicate which granule sizes are supported for stage 2 translation.

If FEAT_GTG is not implemented, ID_AA64MMFR0_EL1.{TGran4, TGran16, TGran64} indicate which granule sizes are supported.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SL0, bits [7:6]

When FEAT_TTST is implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == 0):

Starting level of the stage 2 translation lookup, controlled by VTCR_EL2. The meaning of this field depends on the value of VTCR_EL2.TG0.

SL0	Meaning
0b00	If VTCR_EL2.TG0 is 0b00 (4KB granule): • If FEAT_LPA2 is not implemented, start at level 2. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is 0b0, start at level 2. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is 0b1, start at level -1. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.
0b01	If VTCR_EL2.TG0 is 0b00 (4KB granule): • If FEAT_LPA2 is not implemented, start at level 1. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is 0b0, start at level 1. • If FEAT_LPA2 is implemented, the combination of VTCR_EL2.SL0 == 01 and VTCR_EL2.SL2 == 1 is reserved. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.
0b10	If VTCR_EL2.TG0 is 0b00 (4KB granule): • If FEAT_LPA2 is not implemented, start at level 0. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is 0b0, start at level 0. • If FEAT_LPA2 is implemented, the combination of VTCR_EL2.SL0 == 10 and VTCR_EL2.SL2 == 1 is reserved. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.
0b11	If VTCR_EL2.TG0 is 0b00 (4KB granule): • If FEAT_LPA2 is not implemented, start at level 3. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is 0b0, start at level 3. • If FEAT_LPA2 is implemented, the combination of VTCR_EL2.SL0 == 11 and VTCR_EL2.SL2 == 1 is reserved. If VTCR_EL2.TG0 is 0b10 (16KB granule) and FEAT_LPA2 is implemented, start at level 0.

If this field is programmed to a value that is not consistent with the programming of VTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TTST is not implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == 0):

Starting level of the stage 2 translation lookup, controlled by VTCR_EL2. The meaning of this field depends on the value of VTCR_EL2.TG0.

SL0	Meaning
0b00	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.
0b01	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.
0b10	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by VTTBR_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in 'The AArch64 Virtual Memory System Architecture'.

If this field is programmed to a value that is not consistent with the programming of SL0, then a stage 2 level 0 Translation fault is generated.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, VTCR_EL2.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented, VTCR_EL2.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VTCR_EL2

Unless stated otherwise, any of the bits in VTCR_EL2 are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VTCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x040];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VTCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VTCR_EL2;
```

```
MSR VTCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x040] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VTCR_EL2 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VTTBR_EL2, Virtualization Translation Table Base Register

The VTTBR_EL2 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 2 of an address translation in the EL1&0 translation regime, and other information for this translation regime.

Configuration

AArch64 System register VTTBR_EL2 bits [63:0] are architecturally mapped to AArch32 System register VTTBR[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VTTBR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

VTTBR_EL2 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

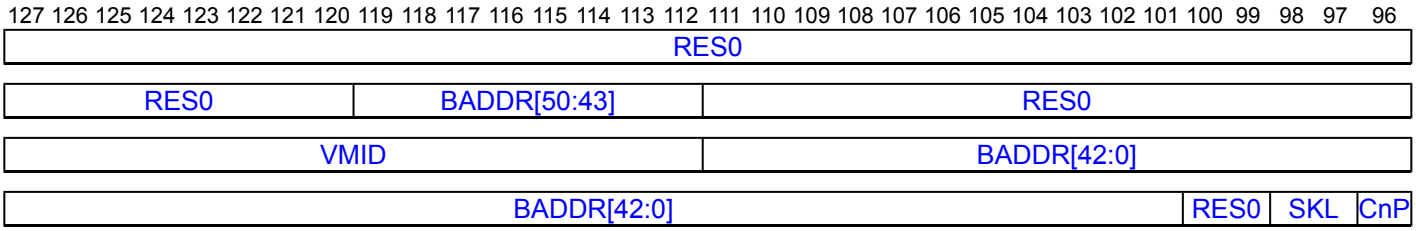
Attributes

VTTBR_EL2 is a:

- 128-bit register when FEAT_D128 is implemented and VTCR_EL2.D128 == 1
- 64-bit register when FEAT_D128 is not implemented or VTCR_EL2.D128 == 0

Field descriptions

When FEAT_D128 is implemented and VTCR_EL2.D128 == 1:



Bits [127:88]

Reserved, RES0.

BADDR, bits [87:80, 47:5]

Translation table base address:

- Bits A[55:x] of the stage 2 translation table base address bits are in register bits[87:80, 47:x].
- Bits A[(x-1):0] of the stage 2 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The BADDR field is split as follows:

- BADDR[50:43] is VTTBR_EL2[87:80].
- BADDR[42:0] is VTTBR_EL2[47:5].

The reset behavior of this field is:

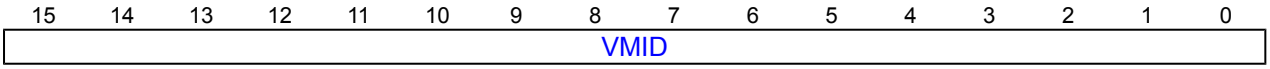
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [79:64]

Reserved, RES0.

VMID, bits [63:48]

VMID encoding when FEAT_VMID16 is implemented and VTCR_EL2.VS == 1



VMID, bits [15:0]

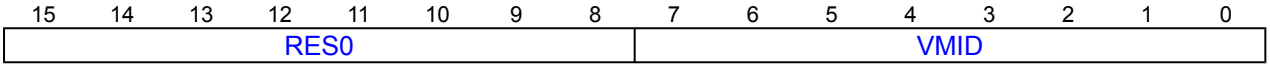
The VMID for the translation table.

If the implementation has an 8-bit VMID, bits [15:8] of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VMID encoding when FEAT_VMID16 is not implemented or VTCR_EL2.VS == 0



Bits [15:8]

Reserved, RES0.

VMID, bits [7:0]

The VMID for the translation table.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level. Skip Level determines the number of levels to be skipped from the regular start level of the Non-Secure stage 2 translation table walk.

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by VTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by VTTBR_EL2 are permitted to differ from the entries for VTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
0b1	The translation table entries pointed to by VTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.

This bit is permitted to be cached in a TLB.

Note

If the value of VTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VTTBR_EL2s do not point to the same translation table entries when using the current VMID then the results of translations using VTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

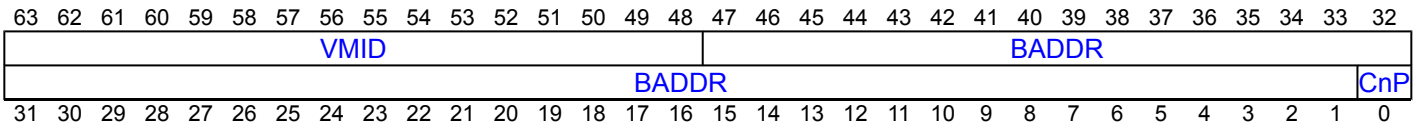
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When FEAT_D128 is not implemented or VTCR_EL2.D128 == 0:



VMID, bits [63:48]

VMID encoding when FEAT_VMID16 is implemented and VTCR_EL2.VS == 1

VMID, bits [15:0]

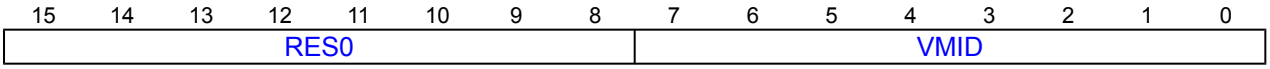
The VMID for the translation table.

If the implementation has an 8-bit VMID, bits [15:8] of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VMID encoding when FEAT_VMID16 is not implemented or VTCR_EL2.VS == 0



Bits [15:8]

Reserved, RES0.

VMID, bits [7:0]

The VMID for the translation table.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, A[47:x] or A[51:x], bits[47:1].

The BADDR field represents a 52-bit address if one of the following applies:

- FEAT_LPA is implemented, the 64KB granule size is in use, and the value of VTCR_EL2.PS is `0b110`.
- FEAT_LPA2 is implemented, the 4KB or 16KB granule size is in use, and the Effective value of VTCR_EL2.DS is 1.
- FEAT_D128 is implemented, 56-bit PAs are supported, the 64KB granule size is in use, and the value of VTCR_EL2.D128 is 0.

When VTTBR_EL2.BADDR represents a 52-bit addresses, all of the following apply:

- Register bits[47:x] hold bits[47:x] of the stage 2 translation table base address, where x is determined by the size of the translation table at the start level.
- The smallest permitted value of x is 6.
- Register bits[5:2] hold bits[51:48] of the stage 2 translation table base address.
- Bits[x:0] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.

For the 64KB granule, if FEAT_LPA is not implemented, and the value of VTCR_EL2.PS is `0b110`, one the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits [55:52] of the stage 2 translation table base address are zero.

If the Effective value of VTCR_EL2.PS is not 0b110 then:

- Register bits[47:x] hold bits[47:x] of the stage 2 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs then bits[51:48] of the translation table base addresses used in this stage of translation are 0b0000.

If any VTTBR_EL2[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using VTTBR_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of VTCR_EL2.T0SZ, the stage of translation, and the translation granule size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]
When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by VTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by VTTBR_EL2 are permitted to differ from the entries for VTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
0b1	The translation table entries pointed to by VTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.

This bit is permitted to be cached in a TLB.

Note

If the value of VTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VTTBR_EL2s do not point to the same translation table entries when using the current VMID then the results of translations using VTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing VTTBR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VTTBR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x020];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VTTBR_EL2<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VTTBR_EL2<63:0>;
```

```
MSR VTTBR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x020] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTTBR_EL2<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    VTTBR_EL2<63:0> = X[t, 64];
```

When FEAT_D128 is implemented

```
MRRS <Xt>, <Xt+1>, VTTBR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, t2, 128] = NVMem[0x020, 128];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = VTTBR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = VTTBR_EL2;
```

When FEAT_D128 is implemented

```
MSRR VTTBR_EL2, <Xt>, <Xt+1>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x020, 128] = X[t, t2, 128];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        VTTBR_EL2<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    VTTBR_EL2<127:0> = X[t, t2, 128];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ZCR_EL1, SVE Control Register (EL1)

The ZCR_EL1 characteristics are:

Purpose

This register controls aspects of SVE visible at Exception levels EL1 and EL0.

Configuration

This register is present only when FEAT_SVE is implemented. Otherwise, direct accesses to ZCR_EL1 are UNDEFINED.

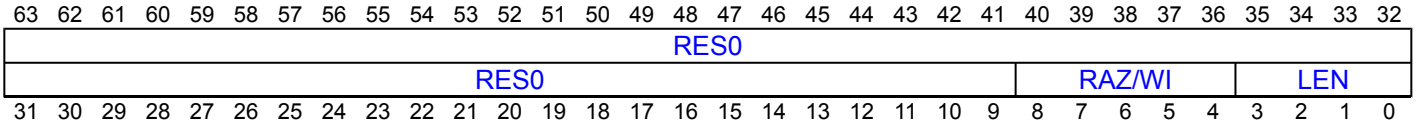
This register has no effect when FEAT_SME is implemented and the PE is in Streaming SVE mode.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this register has no effect on execution at EL0.

Attributes

ZCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Non-streaming SVE vector length at EL1 of (LEN+1)*128 bits. This field also defines the Effective Non-streaming SVE vector length at EL0 when EL2 is not implemented, or EL2 is not enabled in the current Security state, or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

The Non-streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support a subset of the architecturally permitted lengths. An implementation is required to support all lengths that are powers of two, from 128 bits up to its maximum implemented Non-streaming SVE vector length.

When FEAT_SME is not implemented, or the PE is not in Streaming SVE mode, the Effective SVE vector length (VL) is equal to the Effective Non-streaming SVE vector length.

When FEAT_SME is implemented and the PE is in Streaming SVE mode, VL is equal to the Effective Streaming SVE vector length. See SMCR_EL1.

For all purposes other than returning the result of a direct read of ZCR_EL1, the PE selects the Effective Non-streaming SVE vector length by performing checks in the following order:

1. If EL2 is implemented and enabled in the current Security state, and the requested length is greater than the Effective length at EL2, then the Effective length at EL2 is used.
2. If EL3 is implemented and the requested length is greater than the Effective length at EL3, then the Effective length at EL3 is used.

3. Otherwise, the Effective length is the highest supported Non-streaming SVE vector length that is less than or equal to the requested length.

An indirect read of ZCR_EL1.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ZCR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name ZCR_EL1 or ZCR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ZCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x1E0];
    else
        X[t, 64] = ZCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif ELIsInHost(EL2) then
        X[t, 64] = ZCR_EL2;
    else
        X[t, 64] = ZCR_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        X[t, 64] = ZCR_EL1;
```

```
MSR ZCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x1E0] = X[t, 64];
    else
        ZCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif ELIsInHost(EL2) then
        ZCR_EL2 = X[t, 64];
    else
        ZCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented

```
MRS <Xt>, ZCR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x1E0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
            UNDEFINED;
        elsif CPTR_EL2.ZEN IN {'x0'} then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x19);
        else
            X[t, 64] = ZCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            X[t, 64] = ZCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented

```
MSR ZCR_EL12, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x1E0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
            UNDEFINED;
        elsif CPTR_EL2.ZEN IN {'x0'} then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x19);
        else
            ZCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            ZCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ZCR_EL2, SVE Control Register (EL2)

The ZCR_EL2 characteristics are:

Purpose

This register controls aspects of SVE visible at Exception levels EL2, EL1, and EL0.

Configuration

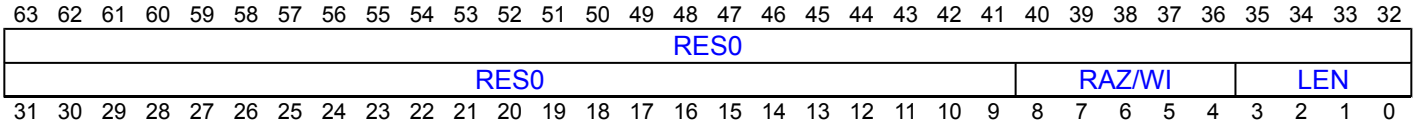
This register is present only when FEAT_SVE is implemented. Otherwise, direct accesses to ZCR_EL2 are UNDEFINED.

This register has no effect when EL2 is not enabled in the current Security state, or when FEAT_SME is implemented and the PE is in Streaming SVE mode.

Attributes

ZCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Non-streaming SVE vector length at EL2 of (LEN+1)*128 bits. This field also defines the Effective Non-streaming SVE vector length at EL0 when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Non-streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support a subset of the architecturally permitted lengths. An implementation is required to support all lengths that are powers of two, from 128 bits up to its maximum implemented Non-streaming SVE vector length.

When FEAT_SME is not implemented, or the PE is not in Streaming SVE mode, the Effective SVE vector length (VL) is equal to the Effective Non-streaming SVE vector length.

When FEAT_SME is implemented and the PE is in Streaming SVE mode, VL is equal to the Effective Streaming SVE vector length. See SMCR_EL2.

For all purposes other than returning the result of a direct read of ZCR_EL2, the PE selects the Effective Non-streaming SVE vector length by performing checks in the following order:

1. If EL3 is implemented and the requested length is greater than the Effective length at EL3, then the Effective length at EL3 is used.
2. Otherwise, the Effective length is the highest supported Non-streaming SVE vector length that is less than or equal to the requested length.

An indirect read of ZCR_EL2.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ZCR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name ZCR_EL2 or ZCR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ZCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    else
        X[t, 64] = ZCR_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        X[t, 64] = ZCR_EL2;
```

```
MSR ZCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL2 = X[t, 64];
```

MRS <Xt>, ZCR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x1E0];
    else
        X[t, 64] = ZCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif ELIsInHost(EL2) then
        X[t, 64] = ZCR_EL2;
    else
        X[t, 64] = ZCR_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        X[t, 64] = ZCR_EL1;
```

MSR ZCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x1E0] = X[t, 64];
    else
        ZCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif ELIsInHost(EL2) then
        ZCR_EL2 = X[t, 64];
    else
        ZCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL1 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ZCR_EL3, SVE Control Register (EL3)

The ZCR_EL3 characteristics are:

Purpose

This register controls aspects of SVE visible at all Exception levels.

Configuration

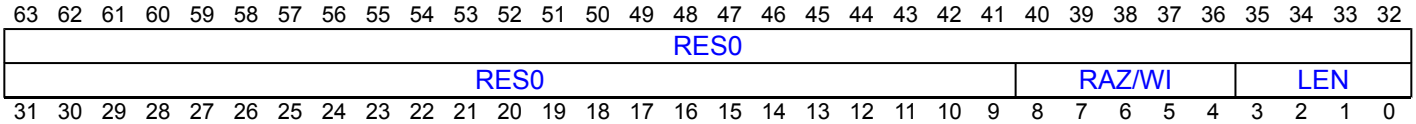
This register is present only when FEAT_SVE is implemented. Otherwise, direct accesses to ZCR_EL3 are UNDEFINED.

This register has no effect when FEAT_SME is implemented and the PE is in Streaming SVE mode.

Attributes

ZCR_EL3 is a 64-bit register.

Field descriptions



Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Non-streaming SVE vector length at EL3 of (LEN+1)*128 bits.

The Non-streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support a subset of the architecturally permitted lengths. An implementation is required to support all lengths that are powers of two, from 128 bits up to its maximum implemented Non-streaming SVE vector length.

When FEAT_SME is not implemented, or the PE is not in Streaming SVE mode, the Effective SVE vector length (VL) is equal to the Effective Non-streaming SVE vector length.

When FEAT_SME is implemented and the PE is in Streaming SVE mode, VL is equal to the Effective Streaming SVE vector length. See SMCR_EL3.

For all purposes other than returning the result of a direct read of ZCR_EL3, the PE selects the highest supported Non-streaming SVE vector length that is less than or equal to the requested length.

An indirect read of ZCR_EL3.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ZCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ZCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        X[t, 64] = ZCR_EL3;
```

```
MSR ZCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL3 = X[t, 64];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AArch32 Registers

ACTLR: Auxiliary Control Register

ACTLR2: Auxiliary Control Register 2

ADFSR: Auxiliary Data Fault Status Register

AIDR: Auxiliary ID Register

AIFSR: Auxiliary Instruction Fault Status Register

AMAIR0: Auxiliary Memory Attribute Indirection Register 0

AMAIR1: Auxiliary Memory Attribute Indirection Register 1

AMCFGR: Activity Monitors Configuration Register

AMCGCR: Activity Monitors Counter Group Configuration Register

AMCNTENCLR0: Activity Monitors Count Enable Clear Register 0

AMCNTENCLR1: Activity Monitors Count Enable Clear Register 1

AMCNTENSET0: Activity Monitors Count Enable Set Register 0

AMCNTENSET1: Activity Monitors Count Enable Set Register 1

AMCR: Activity Monitors Control Register

AMEVCNTR0<n>: Activity Monitors Event Counter Registers 0

AMEVCNTR1<n>: Activity Monitors Event Counter Registers 1

AMEVTYPER0<n>: Activity Monitors Event Type Registers 0

AMEVTYPER1<n>: Activity Monitors Event Type Registers 1

AMUSERENR: Activity Monitors User Enable Register

APSR: Application Program Status Register

CCSIDR: Current Cache Size ID Register

CCSIDR2: Current Cache Size ID Register 2

CLIDR: Cache Level ID Register

CNTFRQ: Counter-timer Frequency register

CNTHCTL: Counter-timer Hyp Control register

CNTHPS_CTL: Counter-timer Secure Physical Timer Control Register (EL2)

CNTHPS_CVAL: Counter-timer Secure Physical Timer CompareValue Register (EL2)

CNTHPS_TVAL: Counter-timer Secure Physical Timer TimerValue Register (EL2)

CNTHP_CTL: Counter-timer Hyp Physical Timer Control register

CNTHP_CVAL: Counter-timer Hyp Physical CompareValue register

CNTHP_TVAL: Counter-timer Hyp Physical Timer TimerValue register

CNTHVS_CTL: Counter-timer Secure Virtual Timer Control Register (EL2)

CNTHVS_CVAL: Counter-timer Secure Virtual Timer CompareValue Register (EL2)

CNTHVS_TVAL: Counter-timer Secure Virtual Timer TimerValue Register (EL2)

CNTHV_CTL: Counter-timer Virtual Timer Control register (EL2)

CNTHV_CVAL: Counter-timer Virtual Timer CompareValue register (EL2)

CNTHV_TVAL: Counter-timer Virtual Timer TimerValue register (EL2)

CNTKCTL: Counter-timer Kernel Control register

CNTPCT: Counter-timer Physical Count register

CNTPCTSS: Counter-timer Self-Synchronized Physical Count register

CNTP_CTL: Counter-timer Physical Timer Control register

CNTP_CVAL: Counter-timer Physical Timer CompareValue register

CNTP_TVAL: Counter-timer Physical Timer TimerValue register

CNTVCT: Counter-timer Virtual Count register

CNTVCTSS: Counter-timer Self-Synchronized Virtual Count register

CNTVOFF: Counter-timer Virtual Offset register

CNTV_CTL: Counter-timer Virtual Timer Control register

CNTV_CVAL: Counter-timer Virtual Timer CompareValue register

CNTV_TVAL: Counter-timer Virtual Timer TimerValue register

CONTEXTIDR: Context ID Register

CPACR: Architectural Feature Access Control Register

CPSR: Current Program Status Register

CSSELR: Cache Size Selection Register

CTR: Cache Type Register

DACR: Domain Access Control Register

DBGAUTHSTATUS: Debug Authentication Status register

DBGBCR<n>: Debug Breakpoint Control Registers

DBGBVR<n>: Debug Breakpoint Value Registers

DBGBXVR<n>: Debug Breakpoint Extended Value Registers

DBGCLAIMCLR: Debug CLAIM Tag Clear register

DBGCLAIMSET: Debug CLAIM Tag Set register

DBGDCCINT: DCC Interrupt Enable Register

DBGDEVID: Debug Device ID register 0

DBGDEVID1: Debug Device ID register 1

DBGDEVID2: Debug Device ID register 2

DBGDIDR: Debug ID Register

DBGDRAR: Debug ROM Address Register

DBGDSAR: Debug Self Address Register

DBGDSCRext: Debug Status and Control Register, External View

DBGDSCRint: Debug Status and Control Register, Internal View

DBGDTRRXext: Debug OS Lock Data Transfer Register, Receive, External View

DBGDTRRXint: Debug Data Transfer Register, Receive

DBGDTRTXext: Debug OS Lock Data Transfer Register, Transmit

DBGDTRTXint: Debug Data Transfer Register, Transmit

DBGOSDLR: Debug OS Double Lock Register

DBGOSECCR: Debug OS Lock Exception Catch Control Register

DBGOSLAR: Debug OS Lock Access Register

DBGOSLSR: Debug OS Lock Status Register

DBGPRCR: Debug Power Control Register

DBGVCR: Debug Vector Catch Register

DBGWCR<n>: Debug Watchpoint Control Registers

DBGWFAR: Debug Watchpoint Fault Address Register

DBGWVR<n>: Debug Watchpoint Value Registers

DFAR: Data Fault Address Register

DFSR: Data Fault Status Register

DISR: Deferred Interrupt Status Register

DLR: Debug Link Register

DSPSR: Debug Saved Program Status Register

DSPSR2: Debug Saved Process State Register 2

ELR_hyp: Exception Link Register (Hyp mode)

ERRIDR: Error Record ID Register

ERRSELR: Error Record Select Register

ERXADDR: Selected Error Record Address Register

ERXADDR2: Selected Error Record Address Register 2

ERXCTLR: Selected Error Record Control Register

ERXCTLR2: Selected Error Record Control Register 2

ERXFR: Selected Error Record Feature Register

ERXFR2: Selected Error Record Feature Register 2

ERXMISC0: Selected Error Record Miscellaneous Register 0

ERXMISC1: Selected Error Record Miscellaneous Register 1

ERXMISC2: Selected Error Record Miscellaneous Register 2

ERXMISC3: Selected Error Record Miscellaneous Register 3

ERXMISC4: Selected Error Record Miscellaneous Register 4

ERXMISC5: Selected Error Record Miscellaneous Register 5

ERXMISC6: Selected Error Record Miscellaneous Register 6

ERXMISC7: Selected Error Record Miscellaneous Register 7

ERXSTATUS: Selected Error Record Primary Status Register

FCSEIDR: FCSE Process ID register

FPEXC: Floating-Point Exception Control register

FPSCR: Floating-Point Status and Control Register

FPSID: Floating-Point System ID register

HACR: Hyp Auxiliary Configuration Register

HACTLR: Hyp Auxiliary Control Register

HACTLR2: Hyp Auxiliary Control Register 2

HADFSR: Hyp Auxiliary Data Fault Status Register

HAIFSR: Hyp Auxiliary Instruction Fault Status Register

HAMAIR0: Hyp Auxiliary Memory Attribute Indirection Register 0

HAMAIR1: Hyp Auxiliary Memory Attribute Indirection Register 1

HCPTR: Hyp Architectural Feature Trap Register

HCR: Hyp Configuration Register

HCR2: Hyp Configuration Register 2

HDCR: Hyp Debug Control Register

HDFAR: Hyp Data Fault Address Register

HIFAR: Hyp Instruction Fault Address Register

HMAIR0: Hyp Memory Attribute Indirection Register 0

HMAIR1: Hyp Memory Attribute Indirection Register 1

HPFAR: Hyp IPA Fault Address Register

HRMR: Hyp Reset Management Register

HSCTLR: Hyp System Control Register

HSR: Hyp Syndrome Register

HSTR: Hyp System Trap Register

HTCR: Hyp Translation Control Register

HTPIDR: Hyp Software Thread ID Register

HTRFCR: Hyp Trace Filter Control Register

HTTBR: Hyp Translation Table Base Register

HVBAR: Hyp Vector Base Address Register

ICC_AP0R<n>: Interrupt Controller Active Priorities Group 0 Registers

ICC_AP1R<n>: Interrupt Controller Active Priorities Group 1 Registers

ICC_ASGI1R: Interrupt Controller Alias Software Generated Interrupt Group 1 Register

ICC_BPR0: Interrupt Controller Binary Point Register 0

ICC_BPR1: Interrupt Controller Binary Point Register 1

ICC_CTLR: Interrupt Controller Control Register

ICC_DIR: Interrupt Controller Deactivate Interrupt Register

ICC_EOIR0: Interrupt Controller End Of Interrupt Register 0

ICC_EOIR1: Interrupt Controller End Of Interrupt Register 1

ICC_HPPIR0: Interrupt Controller Highest Priority Pending Interrupt Register 0

ICC_HPPIR1: Interrupt Controller Highest Priority Pending Interrupt Register 1

ICC_HSRE: Interrupt Controller Hyp System Register Enable register

ICC_IAR0: Interrupt Controller Interrupt Acknowledge Register 0

ICC_IAR1: Interrupt Controller Interrupt Acknowledge Register 1

ICC_IGRPEN0: Interrupt Controller Interrupt Group 0 Enable register

ICC_IGRPEN1: Interrupt Controller Interrupt Group 1 Enable register

ICC_MCTLR: Interrupt Controller Monitor Control Register

ICC_MGRPEN1: Interrupt Controller Monitor Interrupt Group 1 Enable register

ICC_MSRE: Interrupt Controller Monitor System Register Enable register

ICC_PMR: Interrupt Controller Interrupt Priority Mask Register

ICC_RPR: Interrupt Controller Running Priority Register

ICC_SGI0R: Interrupt Controller Software Generated Interrupt Group 0 Register

ICC_SGI1R: Interrupt Controller Software Generated Interrupt Group 1 Register

ICC_SRE: Interrupt Controller System Register Enable register

ICH_AP0R<n>: Interrupt Controller Hyp Active Priorities Group 0 Registers

ICH_AP1R<n>: Interrupt Controller Hyp Active Priorities Group 1 Registers

ICH_EISR: Interrupt Controller End of Interrupt Status Register

ICH_ELRSR: Interrupt Controller Empty List Register Status Register

ICH_HCR: Interrupt Controller Hyp Control Register

ICH_LR<n>: Interrupt Controller List Registers

ICH_LRC<n>: Interrupt Controller List Registers

ICH_MISR: Interrupt Controller Maintenance Interrupt State Register

ICH_VMCR: Interrupt Controller Virtual Machine Control Register

ICH_VTR: Interrupt Controller VGIC Type Register

ICV_AP0R<n>: Interrupt Controller Virtual Active Priorities Group 0 Registers

ICV_AP1R<n>: Interrupt Controller Virtual Active Priorities Group 1 Registers

ICV_BPR0: Interrupt Controller Virtual Binary Point Register 0

ICV_BPR1: Interrupt Controller Virtual Binary Point Register 1

ICV_CTLR: Interrupt Controller Virtual Control Register

ICV_DIR: Interrupt Controller Deactivate Virtual Interrupt Register

ICV_EOIR0: Interrupt Controller Virtual End Of Interrupt Register 0

ICV_EOIR1: Interrupt Controller Virtual End Of Interrupt Register 1

ICV_HPPIR0: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

ICV_HPPIR1: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

ICV_IAR0: Interrupt Controller Virtual Interrupt Acknowledge Register 0

ICV_IAR1: Interrupt Controller Virtual Interrupt Acknowledge Register 1

ICV_IGRPEN0: Interrupt Controller Virtual Interrupt Group 0 Enable register

ICV_IGRPEN1: Interrupt Controller Virtual Interrupt Group 1 Enable register

ICV_PMR: Interrupt Controller Virtual Interrupt Priority Mask Register

ICV_RPR: Interrupt Controller Virtual Running Priority Register

ID_AFR0: Auxiliary Feature Register 0

ID_DFR0: Debug Feature Register 0

ID_DFR1: Debug Feature Register 1

ID_ISAR0: Instruction Set Attribute Register 0

ID_ISAR1: Instruction Set Attribute Register 1

ID_ISAR2: Instruction Set Attribute Register 2

ID_ISAR3: Instruction Set Attribute Register 3

ID_ISAR4: Instruction Set Attribute Register 4

ID_ISAR5: Instruction Set Attribute Register 5

ID_ISAR6: Instruction Set Attribute Register 6

ID_MMFR0: Memory Model Feature Register 0

ID_MMFR1: Memory Model Feature Register 1

ID_MMFR2: Memory Model Feature Register 2

ID_MMFR3: Memory Model Feature Register 3

ID_MMFR4: Memory Model Feature Register 4

ID_MMFR5: Memory Model Feature Register 5

ID_PFR0: Processor Feature Register 0

ID_PFR1: Processor Feature Register 1

ID_PFR2: Processor Feature Register 2

IFAR: Instruction Fault Address Register

IFSR: Instruction Fault Status Register

ISR: Interrupt Status Register

JIDR: Jazelle ID Register

JMCR: Jazelle Main Configuration Register

JOSCR: Jazelle OS Control Register

MAIR0: Memory Attribute Indirection Register 0

MAIR1: Memory Attribute Indirection Register 1

MIDR: Main ID Register

MPIDR: Multiprocessor Affinity Register

MVBAR: Monitor Vector Base Address Register

MVFR0: Media and VFP Feature Register 0

MVFR1: Media and VFP Feature Register 1

MVFR2: Media and VFP Feature Register 2

NMRR: Normal Memory Remap Register

NSACR: Non-Secure Access Control Register

PAR: Physical Address Register

PMCCFILTR: Performance Monitors Cycle Count Filter Register

PMCCNTR: Performance Monitors Cycle Count Register

PMCEID0: Performance Monitors Common Event Identification register 0

PMCEID1: Performance Monitors Common Event Identification register 1

PMCEID2: Performance Monitors Common Event Identification register 2

PMCEID3: Performance Monitors Common Event Identification register 3

PMCNTENCLR: Performance Monitors Count Enable Clear register

PMCNTENSET: Performance Monitors Count Enable Set register

PMCR: Performance Monitors Control Register

PMEVCNTR<n>: Performance Monitors Event Count Registers

PMEVTYPER<n>: Performance Monitors Event Type Registers

PMINTENCLR: Performance Monitors Interrupt Enable Clear register

PMINTENSET: Performance Monitors Interrupt Enable Set register

PMMIR: Performance Monitors Machine Identification Register

PMOVSR: Performance Monitors Overflow Flag Status Register

PMOVSSET: Performance Monitors Overflow Flag Status Set register

PMSELR: Performance Monitors Event Counter Selection Register

PMSWINC: Performance Monitors Software Increment register

PMUSERENR: Performance Monitors User Enable Register

PMXEVCNTR: Performance Monitors Selected Event Count Register

PMXEVTYPER: Performance Monitors Selected Event Type Register

PRRR: Primary Region Remap Register

REVIDR: Revision ID Register

RMR: Reset Management Register

RVBAR: Reset Vector Base Address Register

SCR: Secure Configuration Register

SCTLR: System Control Register

SDCR: Secure Debug Control Register

SDER: Secure Debug Enable Register

SPSR: Saved Program Status Register

SPSR_abt: Saved Program Status Register (Abort mode)

SPSR_fiq: Saved Program Status Register (FIQ mode)

SPSR_hyp: Saved Program Status Register (Hyp mode)

SPSR_irq: Saved Program Status Register (IRQ mode)

SPSR_mon: Saved Program Status Register (Monitor mode)

SPSR_svc: Saved Program Status Register (Supervisor mode)

SPSR_und: Saved Program Status Register (Undefined mode)

TCMTR: TCM Type Register

TLBTR: TLB Type Register

TPIDRPRW: PL1 Software Thread ID Register

TPIDRURO: PL0 Read-Only Software Thread ID Register

TPIDRURW: PL0 Read/Write Software Thread ID Register

TRFCR: Trace Filter Control Register

TTBCR: Translation Table Base Control Register

TTBCR2: Translation Table Base Control Register 2

TTBR0: Translation Table Base Register 0

TTBR1: Translation Table Base Register 1

VBAR: Vector Base Address Register

VDFSR: Virtual SError Exception Syndrome Register

VDISR: Virtual Deferred Interrupt Status Register

VMPIDR: Virtualization Multiprocessor ID Register

VPIDR: Virtualization Processor ID Register

VTCR: Virtualization Translation Control Register

VTTBR: Virtualization Translation Table Base Register

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AArch32 System Instructions

ATS12NSOPR: Address Translate Stages 1 and 2 Non-secure Only PL1 Read

ATS12NSOPW: Address Translate Stages 1 and 2 Non-secure Only PL1 Write

ATS12NSOUR: Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read

ATS12NSOUW: Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write

ATS1CPR: Address Translate Stage 1 Current state PL1 Read

ATS1CPRP: Address Translate Stage 1 Current state PL1 Read PAN

ATS1CPW: Address Translate Stage 1 Current state PL1 Write

ATS1CPWP: Address Translate Stage 1 Current state PL1 Write PAN

ATS1CUR: Address Translate Stage 1 Current state Unprivileged Read

ATS1CUW: Address Translate Stage 1 Current state Unprivileged Write

ATS1HR: Address Translate Stage 1 Hyp mode Read

ATS1HW: Address Translate Stage 1 Hyp mode Write

BPIALL: Branch Predictor Invalidate All

BPIALLIS: Branch Predictor Invalidate All, Inner Shareable

BPIMVA: Branch Predictor Invalidate by VA

CFPRCTX: Control Flow Prediction Restriction by Context

COSPRCTX: Clear Other Speculative Prediction Restriction by Context

CP15DMB: Data Memory Barrier System instruction

CP15DSB: Data Synchronization Barrier System instruction

CP15ISB: Instruction Synchronization Barrier System instruction

CPPRCTX: Cache Prefetch Prediction Restriction by Context

DCCIMVAC: Data Cache line Clean and Invalidate by VA to PoC

DCCISW: Data Cache line Clean and Invalidate by Set/Way

DCCMVAC: Data Cache line Clean by VA to PoC

DCCMVAU: Data Cache line Clean by VA to PoU

DCCSW: Data Cache line Clean by Set/Way

DCIMVAC: Data Cache line Invalidate by VA to PoC

DCISW: Data Cache line Invalidate by Set/Way

DTLBIALL: Data TLB Invalidate All

DTLBIASID: Data TLB Invalidate by ASID match

DTLBIMVA: Data TLB Invalidate by VA

DVPRCTX: Data Value Prediction Restriction by Context

ICIALLU: Instruction Cache Invalidate All to PoU

ICIALLUIS: Instruction Cache Invalidate All to PoU, Inner Shareable

ICIMVAU: Instruction Cache line Invalidate by VA to PoU

ITLBIALL: Instruction TLB Invalidate All

ITLBIASID: Instruction TLB Invalidate by ASID match

ITLBIMVA: Instruction TLB Invalidate by VA

TLBIALL: TLB Invalidate All

TLBIALLH: TLB Invalidate All, Hyp mode

TLBIALLHIS: TLB Invalidate All, Hyp mode, Inner Shareable

TLBIALLIS: TLB Invalidate All, Inner Shareable

TLBIALLNSNH: TLB Invalidate All, Non-Secure Non-Hyp

TLBIALLNSNHIS: TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable

TLBIASID: TLB Invalidate by ASID match

TLBIASIDIS: TLB Invalidate by ASID match, Inner Shareable

TLBIIPAS2: TLB Invalidate by Intermediate Physical Address, Stage 2

TLBIIPAS2IS: TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable

TLBIIPAS2L: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level

TLBIIPAS2LIS: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable

TLBIMVA: TLB Invalidate by VA

TLBIMVAA: TLB Invalidate by VA, All ASID

TLBIMVAAIS: TLB Invalidate by VA, All ASID, Inner Shareable

TLBIMVAAL: TLB Invalidate by VA, All ASID, Last level

TLBIMVAALIS: TLB Invalidate by VA, All ASID, Last level, Inner Shareable

TLBIMVAH: TLB Invalidate by VA, Hyp mode

TLBIMVAHIS: TLB Invalidate by VA, Hyp mode, Inner Shareable

TLBIMVAIS: TLB Invalidate by VA, Inner Shareable

TLBIMVAL: TLB Invalidate by VA, Last level

TLBIMVALH: TLB Invalidate by VA, Last level, Hyp mode

TLBIMVALHIS: TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable

TLBIMVALIS: TLB Invalidate by VA, Last level, Inner Shareable

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ACTLR, Auxiliary Control Register

The ACTLR characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED configuration and control options for execution at EL1 and EL0.

Configuration

This register is banked between ACTLR and ACTLR_S and ACTLR_NS.

AArch32 System register ACTLR bits [31:0] are architecturally mapped to AArch64 System register ACTLR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ACTLR are UNDEFINED.

Some bits might define global configuration settings, and be common to the Secure and Non-secure instances of the register.

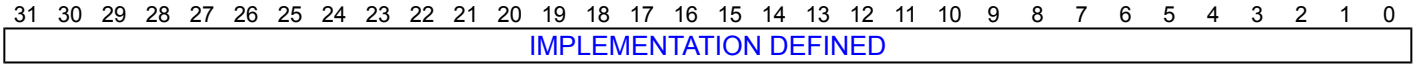
Attributes

ACTLR is a 32-bit register.

This register has the following instances:

- ACTLR, when EL3 is not implemented or FEAT_AA64 is implemented.
- ACTLR_S, when FEAT_AA32EL3 is implemented.
- ACTLR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ACTLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TAC ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = ACTLR_NS;
    else
        R[t] = ACTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = ACTLR_NS;
    else
        R[t] = ACTLR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = ACTLR_S;
    else
        R[t] = ACTLR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TAC ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        ACTLR_NS = R[t];
    else
        ACTLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        ACTLR_NS = R[t];
    else
        ACTLR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        ACTLR_S = R[t];
    else
        ACTLR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ACTLR2, Auxiliary Control Register 2

The ACTLR2 characteristics are:

Purpose

Provides additional space to the ACTLR register to hold IMPLEMENTATION DEFINED trap functionality for execution at EL1 and EL0.

Configuration

This register is banked between ACTLR2 and ACTLR2_S and ACTLR2_NS.

AArch32 System register ACTLR2 bits [31:0] are architecturally mapped to AArch64 System register ACTLR_EL1[63:32].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ACTLR2 are UNDEFINED.

In Armv8.0 and Armv8.1, it is IMPLEMENTATION DEFINED whether this register is implemented, or whether it causes UNDEFINED exceptions when accessed. The implementation of this register can be detected by examining ID_MMFR4.AC2.

From Armv8.2 this register must be implemented.

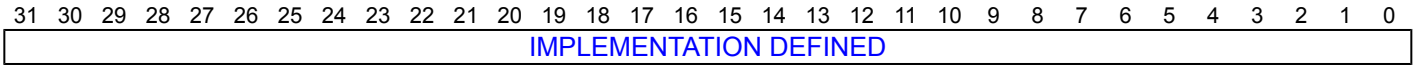
Attributes

ACTLR2 is a 32-bit register.

This register has the following instances:

- ACTLR2, when EL3 is not implemented or FEAT_AA64 is implemented.
- ACTLR2_S, when FEAT_AA32EL3 is implemented.
- ACTLR2_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ACTLR2

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TAC ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = ACTLR2_NS;
    else
        R[t] = ACTLR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = ACTLR2_NS;
    else
        R[t] = ACTLR2;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = ACTLR2_S;
    else
        R[t] = ACTLR2_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TAC ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        ACTLR2_NS = R[t];
    else
        ACTLR2 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        ACTLR2_NS = R[t];
    else
        ACTLR2 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        ACTLR2_S = R[t];
    else
        ACTLR2_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ADFSR, Auxiliary Data Fault Status Register

The ADFSR characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for Data Abort exceptions taken to EL1 modes, and EL3 modes when EL3 is implemented and is using AArch32.

Configuration

This register is banked between ADFSR and ADFSR_S and ADFSR_NS.

AArch32 System register ADFSR bits [31:0] are architecturally mapped to AArch64 System register AFSR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ADFSR are UNDEFINED.

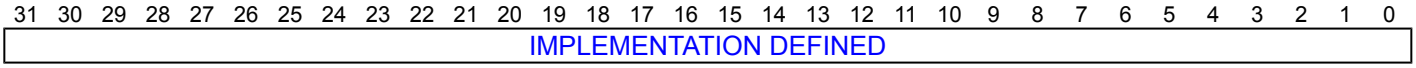
Attributes

ADFSR is a 32-bit register.

This register has the following instances:

- ADFSR, when EL3 is not implemented or FEAT_AA64 is implemented.
- ADFSR_S, when FEAT_AA32EL3 is implemented.
- ADFSR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ADFSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = ADFSR_NS;
    else
        R[t] = ADFSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = ADFSR_NS;
    else
        R[t] = ADFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = ADFSR_S;
    else
        R[t] = ADFSR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        ADFSR_NS = R[t];
    else
        ADFSR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        ADFSR_NS = R[t];
    else
        ADFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        ADFSR_S = R[t];
    else
        ADFSR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AIDR, Auxiliary ID Register

The AIDR characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED identification information.

The value of this register must be used in conjunction with the value of MIDR.

Configuration

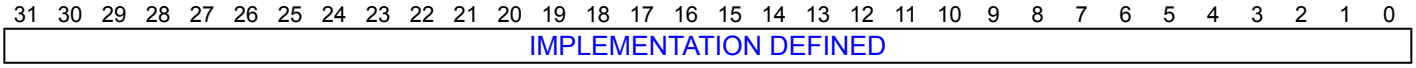
AArch32 System register AIDR bits [31:0] are architecturally mapped to AArch64 System register AIDR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to AIDR are UNDEFINED.

Attributes

AIDR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing AIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b001	0b0000	0b0000	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = AIDR;
elsif PSTATE.EL == EL2 then
    R[t] = AIDR;
elsif PSTATE.EL == EL3 then
    R[t] = AIDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AIFSR, Auxiliary Instruction Fault Status Register

The AIFSR characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for Prefetch Abort exceptions taken to EL1 modes, and EL3 modes when EL3 is implemented and is using AArch32.

Configuration

This register is banked between AIFSR and AIFSR_S and AIFSR_NS.

AArch32 System register AIFSR bits [31:0] are architecturally mapped to AArch64 System register AFSR1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to AIFSR are UNDEFINED.

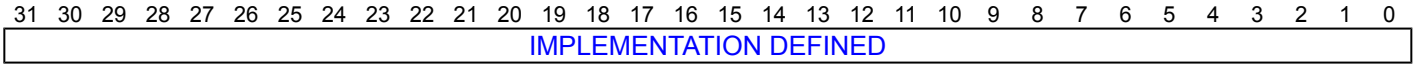
Attributes

AIFSR is a 32-bit register.

This register has the following instances:

- AIFSR, when EL3 is not implemented or FEAT_AA64 is implemented.
- AIFSR_S, when FEAT_AA32EL3 is implemented.
- AIFSR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AIFSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = AIFSR_NS;
    else
        R[t] = AIFSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = AIFSR_NS;
    else
        R[t] = AIFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = AIFSR_S;
    else
        R[t] = AIFSR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        AIFSR_NS = R[t];
    else
        AIFSR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        AIFSR_NS = R[t];
    else
        AIFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AIFSR_S = R[t];
    else
        AIFSR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR0, Auxiliary Memory Attribute Indirection Register 0

The AMAIR0 characteristics are:

Purpose

When using the Long-descriptor format translation tables for stage 1 translations, provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR0.

Configuration

This register is banked between AMAIR0 and AMAIR0_S and AMAIR0_NS.

AArch32 System register AMAIR0 bits [31:0] are architecturally mapped to AArch64 System register AMAIR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to AMAIR0 are UNDEFINED.

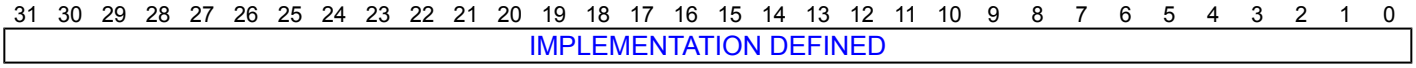
Attributes

AMAIR0 is a 32-bit register.

This register has the following instances:

- AMAIR0, when EL3 is not implemented or FEAT_AA64 is implemented.
- AMAIR0_S, when FEAT_AA32EL3 is implemented.
- AMAIR0_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



This register is RES0 in the following cases:

- When an implementation does not provide any IMPLEMENTATION DEFINED memory attributes.
- When the Long-descriptor translation table format is not used.

If EL3 is implemented and is using AArch32:

- AMAIR0(S) gives the value for memory accesses from Secure state.
- AMAIR0(NS) gives the value for memory accesses from Non-secure states other than Hyp mode.

Any IMPLEMENTATION DEFINED memory attributes are additional qualifiers for the memory locations and must not change the architected behavior specified by MAIR0 and MAIR1.

In a typical implementation, AMAIR0 and AMAIR1 split into eight one-byte fields, corresponding to the MAIRn.Attr<n> fields, but the architecture does not require them to do so.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = AMAIR0_NS;
    else
        R[t] = AMAIR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = AMAIR0_NS;
    else
        R[t] = AMAIR0;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = AMAIR0_S;
    else
        R[t] = AMAIR0_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        AMAIR0_NS = R[t];
    else
        AMAIR0 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        AMAIR0_NS = R[t];
    else
        AMAIR0 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            AMAIR0_S = R[t];
        else
            AMAIR0_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR1, Auxiliary Memory Attribute Indirection Register 1

The AMAIR1 characteristics are:

Purpose

When using the Long-descriptor format translation tables for stage 1 translations, provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR1.

Configuration

This register is banked between AMAIR1 and AMAIR1_S and AMAIR1_NS.

AArch32 System register AMAIR1 bits [31:0] are architecturally mapped to AArch64 System register AMAIR_EL1[63:32].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to AMAIR1 are UNDEFINED.

When EL3 is using AArch32, write access to AMAIR1(S) is disabled when the **CP15SDISABLE** signal is asserted HIGH.

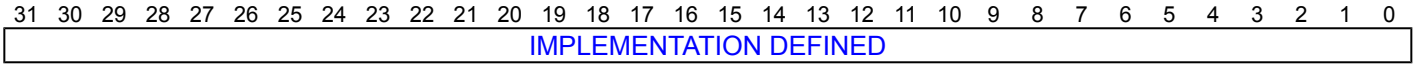
Attributes

AMAIR1 is a 32-bit register.

This register has the following instances:

- AMAIR1, when EL3 is not implemented or FEAT_AA64 is implemented.
- AMAIR1_S, when FEAT_AA32EL3 is implemented.
- AMAIR1_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



This register is RES0 in the following cases:

- When an implementation does not provide any IMPLEMENTATION DEFINED memory attributes.
- When the Long-descriptor translation table format is not used.

If EL3 is implemented and is using AArch32:

- AMAIR1(S) gives the value for memory accesses from Secure state.
- AMAIR1(NS) gives the value for memory accesses from Non-secure states other than Hyp mode.

Any IMPLEMENTATION DEFINED memory attributes are additional qualifiers for the memory locations and must not change the architected behavior specified by MAIR0 and MAIR1.

In a typical implementation, AMAIR0 and AMAIR1 split into eight one-byte fields, corresponding to the MAIRn.Attr<n> fields, but the architecture does not require them to do so.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = AMAIR1_NS;
    else
        R[t] = AMAIR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = AMAIR1_NS;
    else
        R[t] = AMAIR1;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = AMAIR1_S;
    else
        R[t] = AMAIR1_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        AMAIR1_NS = R[t];
    else
        AMAIR1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        AMAIR1_NS = R[t];
    else
        AMAIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            AMAIR1_S = R[t];
        else
            AMAIR1_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCFGR, Activity Monitors Configuration Register

The AMCFGR characteristics are:

Purpose

Global configuration register for the activity monitors.

Provides information on supported features, the number of counter groups implemented, the total number of activity monitor event counters implemented, and the size of the counters. AMCFGR is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch32 System register AMCFGR bits [31:0] are architecturally mapped to AArch64 System register AMCFGR_EL0[31:0].

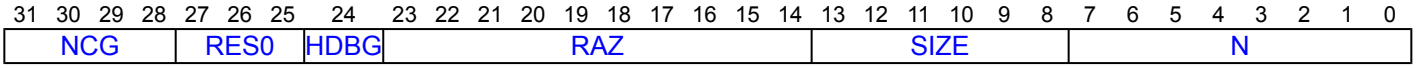
AArch32 System register AMCFGR bits [31:0] are architecturally mapped to External register AMCFGR[31:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMCFGR are UNDEFINED.

Attributes

AMCFGR is a 32-bit register.

Field descriptions



NCG, bits [31:28]

Defines the number of counter groups implemented, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCG	Meaning
0b0000	One counter group implemented.
0b0001	Two counter groups implemented.

All other values are reserved.

Access to this field is **RO**.

Bits [27:25]

Reserved, RES0.

HDBG, bit [24]

Halt-on-debug supported.

This feature must be supported, and so this bit is 0b1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HDBG	Meaning
0b0	AMCR.HDBG is RES0.
0b1	AMCR.HDBG is read/write.

Access to this field is **RO**.

Bits [23:14]

Reserved, RAZ.

SIZE, bits [13:8]

Defines the size of the activity monitor event counters, minus one.

The counters are 64-bit, so the value of this field is `0b111111`.

This field is used by software to determine the spacing of the counters in the memory-map. The counters are at doubleword-aligned addresses.

Reads as `0b111111`.

Access to this field is **RO**.

N, bits [7:0]

Defines the number of activity monitor event counters implemented in all groups, minus one.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMCFGR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCFGR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCFGR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCFGR;
elsif PSTATE.EL == EL3 then
    R[t] = AMCFGR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCGCR, Activity Monitors Counter Group Configuration Register

The AMCGCR characteristics are:

Purpose

Provides information on the number of activity monitor event counters implemented within each counter group.

Configuration

AArch32 System register AMCGCR bits [31:0] are architecturally mapped to AArch64 System register AMCGCR_EL0[31:0].

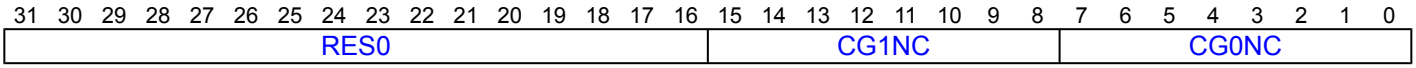
AArch32 System register AMCGCR bits [31:0] are architecturally mapped to External register AMCGCR[31:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMCGCR are UNDEFINED.

Attributes

AMCGCR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

CG1NC, bits [15:8]

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

In an implementation that includes FEAT_AMUv1, the permitted range of values is 0 to 16.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CG0NC, bits [7:0]

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

Reads as `0x04`.

Access to this field is **RO**.

Accessing AMCGCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCGCR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCGCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCGCR;
elsif PSTATE.EL == EL3 then
    R[t] = AMCGCR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENCLR0, Activity Monitors Count Enable Clear Register 0

The AMCNTENCLR0 characteristics are:

Purpose

Disable control bits for the architected activity monitors event counters, AMEVCNTR0<n>.

Configuration

AArch32 System register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR0_EL0[31:0].

AArch32 System register AMCNTENCLR0 bits [31:0] are architecturally mapped to External register AMCNTENCLR0[31:0].

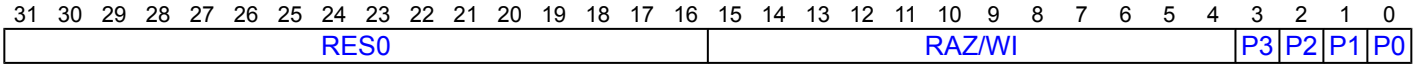
AArch32 System register AMCNTENCLR0 bits [31:0] are architecturally mapped to External register AMCNTENSET0[31:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMCNTENCLR0 are UNDEFINED.

Attributes

AMCNTENCLR0 is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter disable bit for AMEVCNTR0<n>.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR0<n> is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR0<n> is enabled. When written, disables AMEVCNTR0<n>.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= 4, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing AMCNTENCLR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENCLR0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENCLR0;
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENCLR0;
elsif PSTATE.EL == EL3 then
    R[t] = AMCNTENCLR0;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) &&
ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENCLR0 = R[t];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1

The AMCNTENCLR1 characteristics are:

Purpose

Disable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>.

Configuration

AArch32 System register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR1_EL0[31:0].

AArch32 System register AMCNTENCLR1 bits [31:0] are architecturally mapped to External register AMCNTENCLR1[31:0].

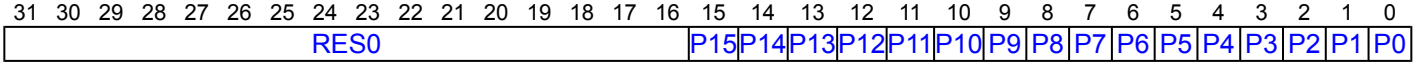
AArch32 System register AMCNTENCLR1 bits [31:0] are architecturally mapped to External register AMCNTENSET1[31:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMCNTENCLR1 are UNDEFINED.

Attributes

AMCNTENCLR1 is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter disable bit for AMEVCNTR1<n>.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR1<n> is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR1<n> is enabled. When written, disables AMEVCNTR1<n>.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= UInt(AMCGCR.CG1NC), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing AMCNTENCLR1

If there are no auxiliary monitor event counters implemented, reads and writes of AMCNTENCLR1 are UNDEFINED.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR.NCG == 0b0000.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENCLR1;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENCLR1;
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENCLR1;
elsif PSTATE.EL == EL3 then
    R[t] = AMCNTENCLR1;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) &&
ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENCLR1 = R[t];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET0, Activity Monitors Count Enable Set Register 0

The AMCNTENSET0 characteristics are:

Purpose

Enable control bits for the architected activity monitors event counters, AMEVCNTR0<n>.

Configuration

AArch32 System register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENSET0_EL0[31:0].

AArch32 System register AMCNTENSET0 bits [31:0] are architecturally mapped to External register AMCNTENSET0[31:0].

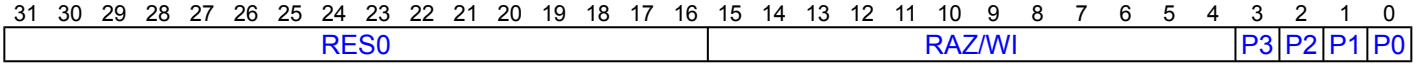
AArch32 System register AMCNTENSET0 bits [31:0] are architecturally mapped to External register AMCNTENCLR0[31:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMCNTENSET0 are UNDEFINED.

Attributes

AMCNTENSET0 is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter enable bit for AMEVCNTR0<n>.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR0<n> is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR0<n> is enabled. When written, enables AMEVCNTR0<n>.

The reset behavior of this field is:

- On an AMU reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When n >= 4, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing AMCNTENSET0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENSET0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENSET0;
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENSET0;
elsif PSTATE.EL == EL3 then
    R[t] = AMCNTENSET0;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) &&
ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENSET0 = R[t];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET1, Activity Monitors Count Enable Set Register 1

The AMCNTENSET1 characteristics are:

Purpose

Enable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>.

Configuration

AArch32 System register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENSET1_EL0[31:0].

AArch32 System register AMCNTENSET1 bits [31:0] are architecturally mapped to External register AMCNTENSET1[31:0].

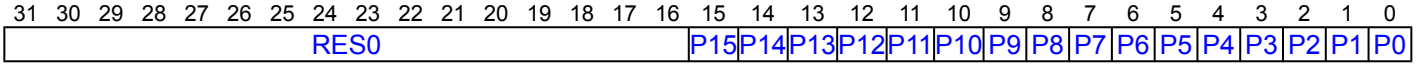
AArch32 System register AMCNTENSET1 bits [31:0] are architecturally mapped to External register AMCNTENCLR1[31:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMCNTENSET1 are UNDEFINED.

Attributes

AMCNTENSET1 is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter enable bit for AMEVCNTR1<n>.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR1<n> is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR1<n> is enabled. When written, enables AMEVCNTR1<n>.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= UInt(AMCGCR.CG1NC), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing AMCNTENSET1

If there are no auxiliary monitor event counters implemented, reads and writes of AMCNTENSET1 are UNDEFINED.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR.NCG == 0b0000.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENSET1;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENSET1;
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCNTENSET1;
elsif PSTATE.EL == EL3 then
    R[t] = AMCNTENSET1;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) &&
ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENSET1 = R[t];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCR, Activity Monitors Control Register

The AMCR characteristics are:

Purpose

Global control register for the activity monitors implementation. AMCR is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch32 System register AMCR bits [31:0] are architecturally mapped to AArch64 System register AMCR_EL0[31:0].

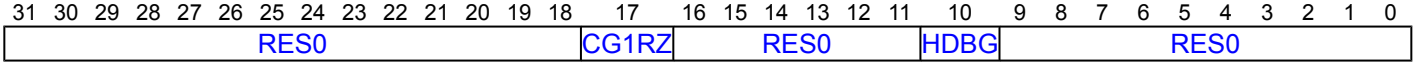
AArch32 System register AMCR bits [31:0] are architecturally mapped to External register AMCR[31:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMCR are UNDEFINED.

Attributes

AMCR is a 32-bit register.

Field descriptions



Bits [31:18]

Reserved, RES0.

CG1RZ, bit [17]

When FEAT_AMUv1p1 is implemented:

Counter Group 1 Read Zero.

CG1RZ	Meaning
0b0	System register reads of AMEVCNTR1<n> return the event count at all implemented and enabled Exception levels.
0b1	If the current Exception level is the highest implemented Exception level, System register reads of AMEVCNTR1<n> return the event count. Otherwise, reads of AMEVCNTR1<n> return a zero value.

Note

Reads from the memory-mapped view are unaffected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:11]

Reserved, RES0.

HDBG, bit [10]

This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

HDBG	Meaning
0b0	Activity monitors do not halt counting when the PE is halted in Debug state.
0b1	Activity monitors halt counting when the PE is halted in Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing AMCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0010	0b000

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMCR;
elsif PSTATE.EL == EL3 then
    R[t] = AMCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0010	0b000

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) &&
ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCR = R[t];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTR0<n>, Activity Monitors Event Counter Registers 0, n = 0 - 3

The AMEVCNTR0<n> characteristics are:

Purpose

Provides access to the architected activity monitor event counters.

Configuration

AArch32 System register AMEVCNTR0<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVCNTR0<n>_EL0[63:0].

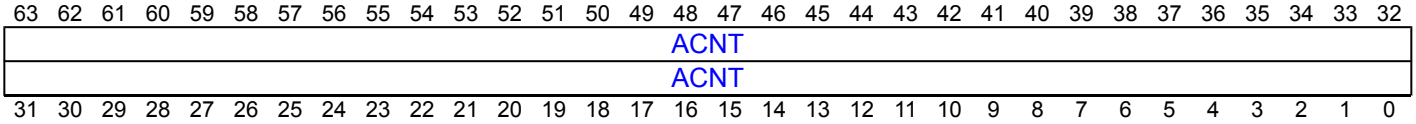
AArch32 System register AMEVCNTR0<n> bits [63:0] are architecturally mapped to External register AMEVCNTR0<n>[63:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMEVCNTR0<n> are UNDEFINED.

Attributes

AMEVCNTR0<n> is a 64-bit register.

Field descriptions



ACNT, bits [63:0]

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 3.

If FEAT_AMUv1p1 is implemented, HCR_EL2.AMVOFFEN is 1, SCR_EL3.AMVOFFEN is 1, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and EL2 is using AArch64 and is implemented in the current Security state, access to these registers at EL0 or EL1 return (PCount<63:0> - AMEVCNTVOFF0<n>_EL2<63:0>).

PCount is the physical count returned when AMEVCNTR0<n> is read from EL2 or EL3.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0000000000000000000000000000000000000000000000000000000000000000'.

Accessing AMEVCNTR0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVCNTR0<n> are UNDEFINED.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm> ; Where m = 0-3
```

coproc	CRm	opc1
0b1111	0b000:m[3]	0b0:m[2:0]

```
integer m = UInt(CRm<0>:opc1<2:0>);

if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && m < 8 && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && m < 8 &&
HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMEVCNTR0<m>_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        R[t, t2] = AMEVCNTR0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && m < 8 &&
HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && m < 8 &&
HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        R[t, t2] = AMEVCNTR0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        R[t, t2] = AMEVCNTR0[m];
elsif PSTATE.EL == EL3 then
    R[t, t2] = AMEVCNTR0[m];
```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm> ; Where m = 0-3

coproc	CRm	opc1
0b1111	0b000:m[3]	0b0:m[2:0]

```
integer m = UInt(CRm<0>:opc1<2:0>);

if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && m < 8 && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) &&
ELUsingAArch32(EL2) && m < 8 && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x04);
elsif IsHighestEL(PSTATE.EL) then
    AMEVCNTR0[m] = R[t2]:R[t];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTR1<n>, Activity Monitors Event Counter Registers 1, n = 0 - 15

The AMEVCNTR1<n> characteristics are:

Purpose

Provides access to the auxiliary activity monitor event counters.

Configuration

AArch32 System register AMEVCNTR1<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVCNTR1<n>_EL0[63:0].

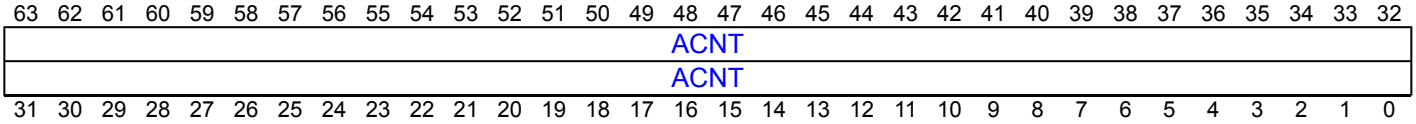
AArch32 System register AMEVCNTR1<n> bits [63:0] are architecturally mapped to External register AMEVCNTR1<n>[63:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMEVCNTR1<n> are UNDEFINED.

Attributes

AMEVCNTR1<n> is a 64-bit register.

Field descriptions



ACNT, bits [63:0]

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If FEAT_AMUv1p1 is implemented, HCR_EL2.AMVOFFEN is 1, SCR_EL3.AMVOFFEN is 1, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, EL2 is using AArch64 and is implemented in the current Security state, and AMCR_EL0.CG1RZ is 0, reads to these registers at EL0 or EL1 return (PCount<63:0> - AMEVCNTVOFF1<n>_EL2<63:0>).

PCount is the physical count returned when AMEVCNTR1<n> is read from EL2 or EL3.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0000000000000000000000000000000000000000000000000000000000000000'.

Accessing AMEVCNTR1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVCNTR1<n> are UNDEFINED.

Note

AMCGCR.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm> ; Where m = 0-15
```

coproc	CRm	opc1
0b1111	0b010:m[3]	0b0:m[2:0]

```
integer m = UInt(CRm<0>:opc1<2:0>);

if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && m >= 8 && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && m >= 8 &&
HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMEVCNTR1<m>_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    elsif IsFeatureImplemented(FEAT_AA64) && AMCR_EL0.CG1RZ == '1' then
        R[t, t2] = Zeros(64);
    elsif !IsFeatureImplemented(FEAT_AA64) && AMCR.CG1RZ == '1' then
        R[t, t2] = Zeros(64);
    else
        R[t, t2] = AMEVCNTR1[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && m >= 8 &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && m >= 8 &&
HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
```

```
== '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    elsif !IsHighestEL(PSTATE.EL) && IsFeatureImplemented(FEAT_AA64) && AMCR_EL0.CG1RZ == '1' then
        R[t, t2] = Zeros(64);
    elsif !IsHighestEL(PSTATE.EL) && !IsFeatureImplemented(FEAT_AA64) && AMCR.CG1RZ == '1' then
        R[t, t2] = Zeros(64);
    else
        R[t, t2] = AMEVCNTR1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    elsif !IsHighestEL(PSTATE.EL) && IsFeatureImplemented(FEAT_AA64) && AMCR_EL0.CG1RZ == '1' then
        R[t, t2] = Zeros(64);
    elsif !IsHighestEL(PSTATE.EL) && !IsFeatureImplemented(FEAT_AA64) && AMCR.CG1RZ == '1' then
        R[t, t2] = Zeros(64);
    else
        R[t, t2] = AMEVCNTR1[m];
elsif PSTATE.EL == EL3 then
    R[t, t2] = AMEVCNTR1[m];
```

`MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm> ; Where m = 0-15`

coproc	CRm	opc1
0b1111	0b010:m[3]	0b0:m[2:0]

```
integer m = UInt(CRm<0>:opc1<2:0>);

if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && m >= 8 && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) &&
ELUsingAArch32(EL2) && m >= 8 && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x04);
elsif IsHighestEL(PSTATE.EL) then
    AMEVCNTR1[m] = R[t2]:R[t];
else
    UNDEFINED;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVTYPER0<n>, Activity Monitors Event Type Registers 0, n = 0 - 3

The AMEVTYPER0<n> characteristics are:

Purpose

Provides information on the events that an architected activity monitor event counter AMEVCNTR0<n> counts.

Configuration

AArch32 System register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPER0<n>_EL0[31:0].

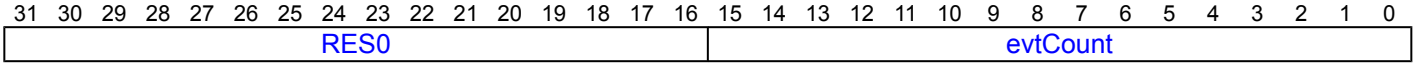
AArch32 System register AMEVTYPER0<n> bits [31:0] are architecturally mapped to External register AMEVTYPER0<n>[31:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMEVTYPER0<n> are UNDEFINED.

Attributes

AMEVTYPER0<n> is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

evtCount	Meaning	Applies when
0x0011	Processor frequency cycles.	When n == 0
0x4004	Constant frequency cycles.	When n == 1
0x0008	Instructions retired.	When n == 2
0x4005	Memory stall cycles.	When n == 3

Access to this field is **RO**.

Accessing AMEVTYPER0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPER0<n> are UNDEFINED.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b011:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMEVTYPER0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMEVTYPER0[m];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMEVTYPER0[m];
elsif PSTATE.EL == EL3 then
    R[t] = AMEVTYPER0[m];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVTYPER1<n>, Activity Monitors Event Type Registers 1, n = 0 - 15

The AMEVTYPER1<n> characteristics are:

Purpose

Provides information on the events that an auxiliary activity monitor event counter AMEVCNTR1<n> counts.

Configuration

AArch32 System register AMEVTYPER1<n> bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPER1<n>_EL0[31:0].

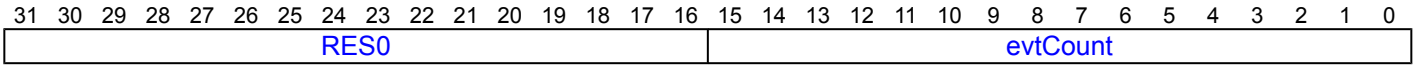
AArch32 System register AMEVTYPER1<n> bits [31:0] are architecturally mapped to External register AMEVTYPER1<n>[31:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMEVTYPER1<n> are UNDEFINED.

Attributes

AMEVTYPER1<n> is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter AMEVCNTR1<n>.

It is IMPLEMENTATION DEFINED what values are supported by each counter.

If software writes a value to this field which is not supported by the corresponding counter AMEVCNTR1<n>, then:

- It is UNPREDICTABLE which event will be counted.
- The value read back is UNKNOWN.

The event counted by AMEVCNTR1<n> might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.

If the corresponding counter AMEVCNTR1<n> is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVTYPER1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVTYPER1<n> are UNDEFINED.

Note

AMCGCR.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b111:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMEVTYPER1<m>_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMEVTYPER1[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMEVTYPER1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMEVTYPER1[m];
elsif PSTATE.EL == EL3 then
    R[t] = AMEVTYPER1[m];
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b111:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) &&
ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) && !boolean IMPLEMENTATION_DEFINED "AMEVCNTR1[m] is fixed" then
    AMEVTYPER1[m] = R[t];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMUSERENR, Activity Monitors User Enable Register

The AMUSERENR characteristics are:

Purpose

Global user enable register for the activity monitors. Enables or disables EL0 access to the activity monitors. AMUSERENR is applicable to both the architected and the auxiliary counter groups.

Configuration

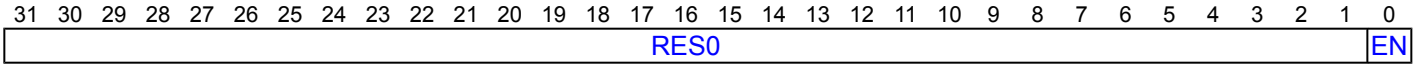
AArch32 System register AMUSERENR bits [31:0] are architecturally mapped to AArch64 System register AMUSERENR_EL0[31:0].

This register is present only when FEAT_AMUv1 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to AMUSERENR are UNDEFINED.

Attributes

AMUSERENR is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

EN, bit [0]

Traps EL0 accesses to the activity monitors registers to EL1.

EN	Meaning
0b0	EL0 accesses to the activity monitors registers are trapped to EL1.
0b1	This control does not cause any instructions to be trapped. Software can access all activity monitor registers at EL0.

Note

- AMUSERENR can always be read at EL0 and is not governed by this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMUSERENR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMUSERENR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMUSERENR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = AMUSERENR;
elsif PSTATE.EL == EL3 then
    R[t] = AMUSERENR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_AMUv1) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TAM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        AMUSERENR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        AMUSERENR = R[t];
elsif PSTATE.EL == EL3 then
    AMUSERENR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APSR, Application Program Status Register

The APSR characteristics are:

Purpose

Hold program status and control information.

Note

Some of the fields in this register are permitted to return the value of the PSTATE field on a read. This is an exception to the general rule that an UNKNOWN field must not return information that cannot be obtained, at the current Privilege level, by an architected mechanism.

For more information see 'The Application Program Status Register, APSR'.

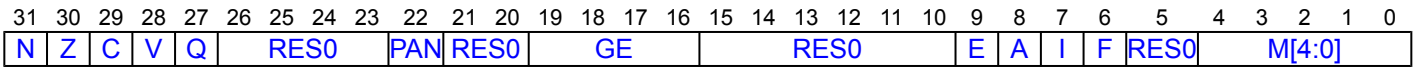
Configuration

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to APSR are UNDEFINED.

Attributes

APSR is a 32-bit register.

Field descriptions



N, bit [31]

Negative condition flag. Set to bit[31] of the result of the last flag-setting instruction. If the result is regarded as a two's complement signed integer, then N is set to 1 if the result was negative, and N is set to 0 if the result was positive or zero.

Z, bit [30]

Zero condition flag. Set to 1 if the result of the last flag-setting instruction was zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

C, bit [29]

Carry condition flag. Set to 1 if the last flag-setting instruction resulted in a carry condition, for example an unsigned overflow on an addition.

V, bit [28]

Overflow condition flag. Set to 1 if the last flag-setting instruction resulted in an overflow condition, for example a signed overflow on an addition.

Q, bit [27]

Cumulative saturation bit. Set to 1 to indicate that overflow or saturation occurred in some instructions.

Bits [26:23]

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. This field is UNKNOWN, but is permitted to return the value of PSTATE.PAN field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [21:20]

Reserved, RES0.

GE, bits [19:16]

Greater than or Equal flags, for parallel addition and subtraction.

Bits [15:10]

Reserved, RES0.

E, bit [9]

Endianness. This field is UNKNOWN, but is permitted to return the value of PSTATE.E field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. This field is UNKNOWN, but is permitted to return the value of PSTATE.A field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. This field is UNKNOWN, but is permitted to return the value of PSTATE.I field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. This field is UNKNOWN, but is permitted to return the value of PSTATE.F field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4:0], bits [4:0]

Mode. This field is UNKNOWN, but is permitted to return the value of PSTATE.M[4:0] field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APSR

APSR can be read using the MRS instruction and written using the MSR (register) or MSR (immediate) instructions.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS12NSOPR, Address Translate Stages 1 and 2 Non-secure Only PL1 Read

The ATS12NSOPR characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL1 and the Non-secure state, with permissions as if reading from the given virtual address.

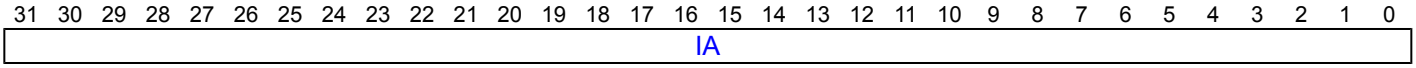
Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to ATS12NSOPR are UNDEFINED.

Attributes

ATS12NSOPR is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing ATS12NSOPR

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1000	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_12, EL1, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_12, EL1, ATAccess_Read);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS12NSOPW, Address Translate Stages 1 and 2 Non-secure Only PL1 Write

The ATS12NSOPW characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL1 and the Non-secure state, with permissions as if writing to the given virtual address.

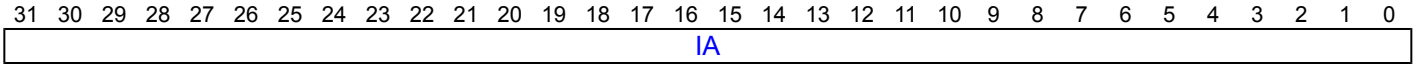
Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to ATS12NSOPW are UNDEFINED.

Attributes

ATS12NSOPW is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing ATS12NSOPW

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1000	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_12, EL1, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_12, EL1, ATAccess_Write);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS12NSOUR, Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read

The ATS12NSOUR characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL0 and the Non-secure state, with permissions as if reading from the given virtual address.

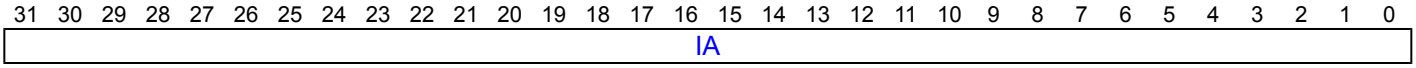
Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to ATS12NSOUR are UNDEFINED.

Attributes

ATS12NSOUR is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing ATS12NSOUR

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1000	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_12, EL0, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_12, EL0, ATAccess_Read);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS12NSOUW, Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write

The ATS12NSOUW characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL0 and the Non-secure state, with permissions as if writing to the given virtual address.

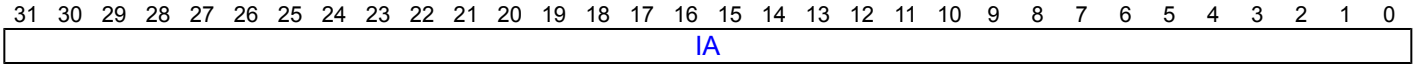
Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to ATS12NSOUW are UNDEFINED.

Attributes

ATS12NSOUW is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing ATS12NSOUW

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1000	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_12, EL0, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_12, EL0, ATAccess_Write);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CPR, Address Translate Stage 1 Current state PL1 Read

The ATS1CPR characteristics are:

Purpose

Performs stage 1 address translation as defined for PL1 and the current Security state, with permissions as if reading from the given virtual address.

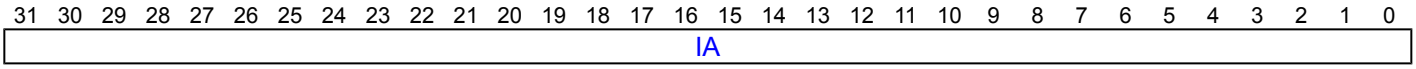
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ATS1CPR are UNDEFINED.

Attributes

ATS1CPR is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing ATS1CPR

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Read);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AArch32.AT(R[t], TranslationStage_1, EL3, ATAccess_Read);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Read);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CPRP, Address Translate Stage 1 Current state PL1 Read PAN

The ATS1CPRP characteristics are:

Purpose

Performs a stage 1 address translation at PL1 and in the current Security state, where the value of PSTATE.PAN determines if a read from a location will generate a Permission fault for a privileged access.

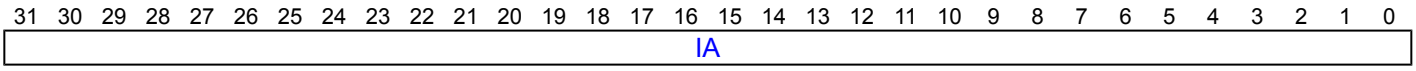
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented and FEAT_PAN2 is implemented. Otherwise, direct accesses to ATS1CPRP are UNDEFINED.

Attributes

ATS1CPRP is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing ATS1CPRP

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1001	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_PAN2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_ReadPAN);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_ReadPAN);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AArch32.AT(R[t], TranslationStage_1, EL3, ATAccess_ReadPAN);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_ReadPAN);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CPW, Address Translate Stage 1 Current state PL1 Write

The ATS1CPW characteristics are:

Purpose

Performs stage 1 address translation as defined for PL1 and the current Security state, with permissions as if writing to the given virtual address.

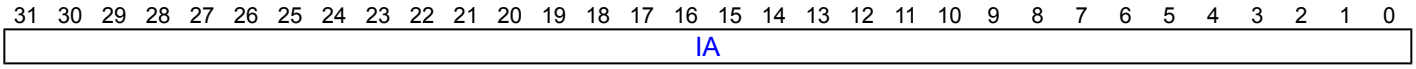
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ATS1CPW are UNDEFINED.

Attributes

ATS1CPW is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing ATS1CPW

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Write);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AArch32.AT(R[t], TranslationStage_1, EL3, ATAccess_Write);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Write);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CPWP, Address Translate Stage 1 Current state PL1 Write PAN

The ATS1CPWP characteristics are:

Purpose

Performs a stage 1 address translation at PL1 and in the current Security state, where the value of PSTATE.PAN determines if a write to the location will generate a Permission fault for a privileged access.

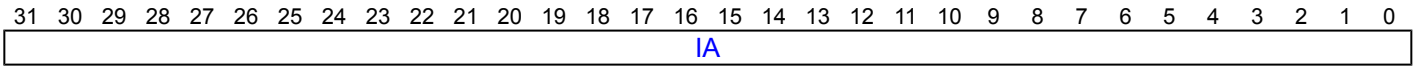
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented and FEAT_PAN2 is implemented. Otherwise, direct accesses to ATS1CPWP are UNDEFINED.

Attributes

ATS1CPWP is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing ATS1CPWP

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1001	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_PAN2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_WritePAN);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_WritePAN);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AArch32.AT(R[t], TranslationStage_1, EL3, ATAccess_WritePAN);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_WritePAN);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CUR, Address Translate Stage 1 Current state Unprivileged Read

The ATS1CUR characteristics are:

Purpose

Performs stage 1 address translation as defined for PL0 and the current Security state, with permissions as if reading from the given virtual address.

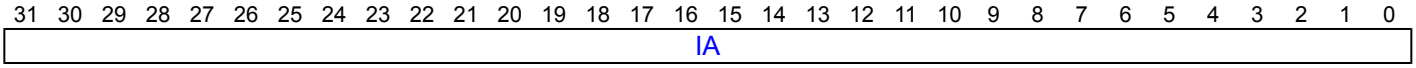
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ATS1CUR are UNDEFINED.

Attributes

ATS1CUR is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing ATS1CUR

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Read);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Read);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CUW, Address Translate Stage 1 Current state Unprivileged Write

The ATS1CUW characteristics are:

Purpose

Performs stage 1 address translation as defined for PL0 and the current Security state, with permissions as if writing to the given virtual address.

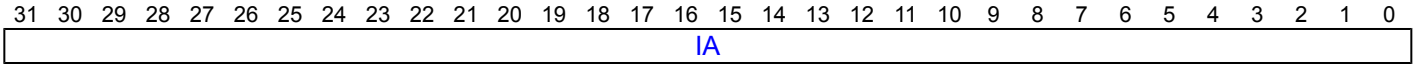
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ATS1CUW are UNDEFINED.

Attributes

ATS1CUW is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing ATS1CUW

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1000	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Write);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Write);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1HR, Address Translate Stage 1 Hyp mode Read

The ATS1HR characteristics are:

Purpose

Performs stage 1 address translation as defined for PL2 and the Non-secure state, with permissions as if reading from the given virtual address.

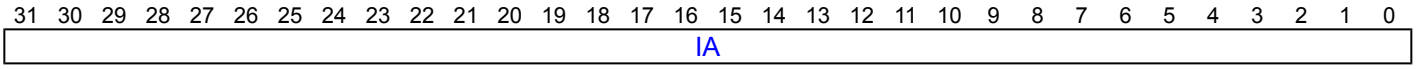
Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to ATS1HR are UNDEFINED.

Attributes

ATS1HR is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the translation.

Executing ATS1HR

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0111	0b1000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL2, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_1, EL2, ATAccess_Read);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1HW, Address Translate Stage 1 Hyp mode Write

The ATS1HW characteristics are:

Purpose

Performs stage 1 address translation as defined for PL2 and the Non-secure state, with permissions as if writing to the given virtual address.

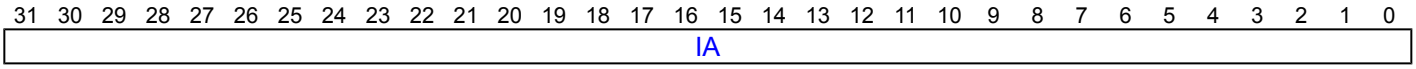
Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to ATS1HW are UNDEFINED.

Attributes

ATS1HW is a 32-bit System instruction.

Field descriptions



IA, bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the translation.

Executing ATS1HW

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0111	0b1000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL2, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_1, EL2, ATAccess_Write);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BPIALL, Branch Predictor Invalidate All

The BPIALL characteristics are:

Purpose

Invalidate all entries from branch predictors.

Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to BPIALL are UNDEFINED.

In an implementation where the branch predictors are architecturally invisible, this instruction can execute as a NOP.

Attributes

BPIALL is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing BPIALL

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

When HCR.FB is 1, at Non-secure EL1 this instruction executes as a BPIALLIS.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0101	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FB ==
'1' then
        BPIALLIS();
    else
        BPIALL();
elsif PSTATE.EL == EL2 then
    BPIALL();
elsif PSTATE.EL == EL3 then
    BPIALL();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BPIALLIS, Branch Predictor Invalidate All, Inner Shareable

The BPIALLIS characteristics are:

Purpose

Invalidate all entries from branch predictors Inner Shareable.

Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to BPIALLIS are UNDEFINED.

In an implementation where the branch predictors are architecturally invisible, this instruction can execute as a NOP.

Attributes

BPIALLIS is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing BPIALLIS

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0001	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        BPIALLIS();
elsif PSTATE.EL == EL2 then
    BPIALLIS();
elsif PSTATE.EL == EL3 then
    BPIALLIS();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BPIMVA, Branch Predictor Invalidate by VA

The BPIMVA characteristics are:

Purpose

Invalidate virtual address from branch predictors.

Configuration

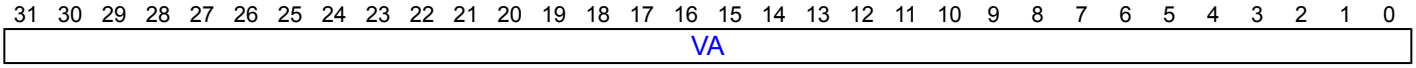
This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to BPIMVA are UNDEFINED.

In an implementation where the branch predictors are architecturally invisible, this instruction can execute as a NOP.

Attributes

BPIMVA is a 32-bit System instruction.

Field descriptions



VA, bits [31:0]

Virtual address to use.

Executing BPIMVA

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0101	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        BPIMVA(R[t]);
elsif PSTATE.EL == EL2 then
    BPIMVA(R[t]);
elsif PSTATE.EL == EL3 then
    BPIMVA(R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CCSIDR, Current Cache Size ID Register

The CCSIDR characteristics are:

Purpose

Provides information about the architecture of the currently selected cache.

When FEAT_CCIDX is implemented, this register is used in conjunction with CCSIDR2.

Configuration

AArch32 System register CCSIDR bits [31:0] are architecturally mapped to AArch64 System register CCSIDR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to CCSIDR are UNDEFINED.

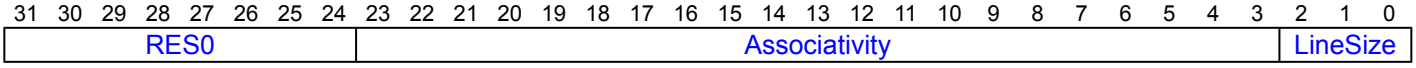
The implementation includes one CCSIDR for each cache that it can access. CSSELR and the Security state select which Cache Size ID Register is accessible.

Attributes

CCSIDR is a 32-bit register.

Field descriptions

When FEAT_CCIDX is implemented:



Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [31:24]

Reserved, RES0.

Associativity, bits [23:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log_2(Number of bytes in cache line)) - 4. For example:

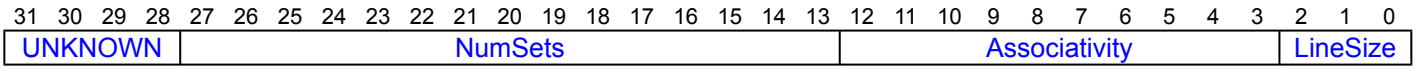
For a line length of 16 bytes: $Log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.

For a line length of 32 bytes: $Log_2(32) = 5$, LineSize entry = 1.

Note

The C++ 17 specification has two defined parameters relating to the granularity of memory that does not interfere. For generic software and tools, Arm will set the hardware_destructive_interference_size parameter to 256 bytes and the hardware_constructive_interference_size parameter to 64 bytes.

Otherwise:



Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [31:28]

Reserved, UNKNOWN.

NumSets, bits [27:13]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Associativity, bits [12:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log_2(Number of bytes in cache line)) - 4. For example:

For a line length of 16 bytes: $Log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.

For a line length of 32 bytes: $Log_2(32) = 5$, LineSize entry = 1.

Note

The C++ 17 specification has two defined parameters relating to the granularity of memory that does not interfere. For generic software and tools, Arm will set the hardware_destructive_interference_size parameter to 256 bytes and the hardware_constructive_interference_size parameter to 64 bytes.

Accessing CCSIDR

If CSSELR.{Level, InD} is programmed to a cache level that is not implemented, then on a read of the CCSIDR the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR read is treated as NOP.
- The CCSIDR read is UNDEFINED.
- The CCSIDR read returns an UNKNOWN value.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b001	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_EVT) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_EVT) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = CCSIDR;
elsif PSTATE.EL == EL2 then
    R[t] = CCSIDR;
elsif PSTATE.EL == EL3 then
    R[t] = CCSIDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CCSIDR2, Current Cache Size ID Register 2

The CCSIDR2 characteristics are:

Purpose

Provides information about the architecture of the currently selected cache.

Configuration

AArch32 System register CCSIDR2 bits [31:0] are architecturally mapped to AArch64 System register CCSIDR2_EL1[31:0].

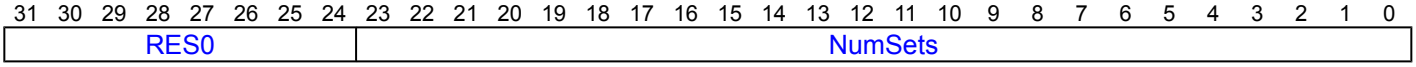
This register is present only when FEAT_AA32EL1 is implemented and FEAT_CCIDX is implemented. Otherwise, direct accesses to CCSIDR2 are UNDEFINED.

The implementation includes one CCSIDR2 for each cache that it can access. CSSELR and the Security state select which Cache Size ID Register is accessible.

Attributes

CCSIDR2 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

NumSets, bits [23:0]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Accessing CCSIDR2

If CSSELR.{Level, InD} is programmed to a cache level that is not implemented, then on a read of the CCSIDR2 the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR2 read is treated as NOP.
- The CCSIDR2 read is UNDEFINED.
- The CCSIDR2 read returns an UNKNOWN value.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b001	0b0000	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_CCIDX)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_EVT) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_EVT) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = CCSIDR2;
elsif PSTATE.EL == EL2 then
    R[t] = CCSIDR2;
elsif PSTATE.EL == EL3 then
    R[t] = CCSIDR2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CFPRCTX, Control Flow Prediction Restriction by Context

The CFPRCTX characteristics are:

Purpose

Control Flow Prediction Restriction by Context applies to all Control Flow Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

Control flow predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control speculative execution occurring after the instruction is complete and synchronized.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

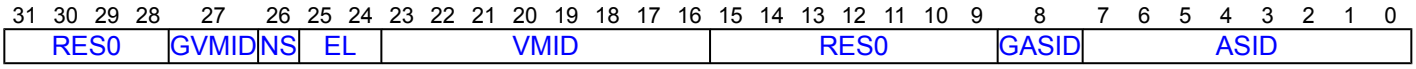
Configuration

This instruction is present only when FEAT_AA32 is implemented and FEAT_SPECRES is implemented. Otherwise, direct accesses to CFPRCTX are UNDEFINED.

Attributes

CFPRCTX is a 32-bit System instruction.

Field descriptions



Bits [31:28]

Reserved, RES0.

GVMID, bit [27]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 target execution context.
0b1	Applies to all VMIDs for an EL0 or EL1 target execution context.

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

NS, bit [26]

Security State.

NS	Meaning
0b0	Secure state.
0b1	Non-secure state.

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

If the instruction is executed at an Exception level lower than the specified level, or is specified to apply to a combination of Exception level and Security state that is not implemented, this instruction is treated as a NOP.

VMID, bits [23:16]

Only applies when bit[27] is 0 and the target execution context is either:

- EL1.
- EL0 when EL2 is using AArch32 or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0, this field is treated as the current VMID if any of the following are true:

- EL2 is using AArch32.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

Bits [15:9]

Reserved, RES0.

GASID, bit [8]

Execution of this instruction applies to all ASIDs or a specified ASID.

GASID	Meaning
0b0	Applies to specified ASID for an EL0 target execution context.
0b1	Applies to all ASIDs for an EL0 target execution context.

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field is treated as 0.

ASID, bits [7:0]

Only applies for an EL0 target execution context and when bit[8] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

Executing CFPRCTX

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0011	0b100

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_SPECRES)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && SCTLR.EnRCTX == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T7
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.CFPRCTX == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnRCTX == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch32.RestrictPrediction(R[t], RestrictType_ControlFlow);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x03);
    else
        AArch32.RestrictPrediction(R[t], RestrictType_ControlFlow);
elsif PSTATE.EL == EL2 then
    AArch32.RestrictPrediction(R[t], RestrictType_ControlFlow);
elsif PSTATE.EL == EL3 then
    AArch32.RestrictPrediction(R[t], RestrictType_ControlFlow);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CLIDR, Cache Level ID Register

The CLIDR characteristics are:

Purpose

Identifies the type of cache, or caches, that are implemented at each level and can be managed using the architected cache maintenance instructions that operate by set/way, up to a maximum of seven levels. Also identifies the Level of Coherence (LoC) and Level of Unification (LoU) for the cache hierarchy.

Configuration

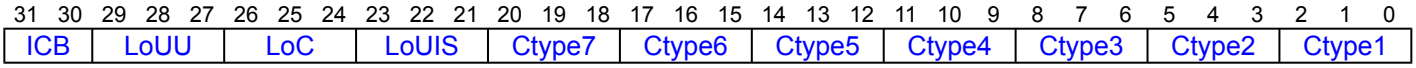
AArch32 System register CLIDR bits [31:0] are architecturally mapped to AArch64 System register CLIDR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to CLIDR are UNDEFINED.

Attributes

CLIDR is a 32-bit register.

Field descriptions



ICB, bits [31:30]

Inner cache boundary. This field indicates the boundary for caching Inner Cacheable memory regions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ICB	Meaning
0b00	Not disclosed by this mechanism.
0b01	L1 cache is the highest Inner Cacheable level.
0b10	L2 cache is the highest Inner Cacheable level.
0b11	L3 cache is the highest Inner Cacheable level.

Access to this field is **RO**.

LoUU, bits [29:27]

Level of Unification Uniprocessor for the cache hierarchy.

For a description of the values of this field, see Terminology for Clean, Invalidate, and Clean and Invalidate instructions.

Note

This field does not describe the requirements for instruction cache invalidation. See CTR.DIC.

Note

When FEAT_S2FWB is implemented, the architecture requires that this field is zero so that no levels of data cache need to be cleaned in order to manage coherency with instruction fetches.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

LoC, bits [26:24]

Level of Coherence for the cache hierarchy.

For a description of the values of this field, see Terminology for Clean, Invalidate, and Clean and Invalidate instructions.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

LoUIS, bits [23:21]

Level of Unification Inner Shareable for the cache hierarchy.

For a description of the values of this field, see Terminology for Clean, Invalidate, and Clean and Invalidate instructions.

Note

This field does not describe the requirements for instruction cache invalidation. See CTR.DIC.

Note

When FEAT_S2FWB is implemented, the architecture requires that this field is zero so that no levels of data cache need to be cleaned in order to manage coherency with instruction fetches.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Ctype<n>, bits [3(n-1)+2:3(n-1)], for n = 7 to 1

Cache Type fields. Indicate the type of cache that is implemented and can be managed using the architected cache maintenance instructions that operate by set/way at each level, from Level 1 up to a maximum of seven levels of cache hierarchy.

Ctype<n>	Meaning
0b000	No cache.
0b001	Instruction cache only.
0b010	Data cache only.
0b011	Separate instruction and data caches.
0b100	Unified cache.

All other values are reserved.

If software reads the Cache Type fields from Ctype1 upwards, once it has seen a value of 0b000, no caches that can be managed using the architected cache maintenance instructions that operate by set/way exist at further-out levels of the hierarchy. So, for example, if Ctype3 is the first Cache Type field with a value of 0b000, the values of Ctype4 to Ctype7 must be ignored.

Accessing CLIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b001	0b0000	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_EVT) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_EVT) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = CLIDR;
elsif PSTATE.EL == EL2 then
    R[t] = CLIDR;
elsif PSTATE.EL == EL3 then
    R[t] = CLIDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTFRQ, Counter-timer Frequency register

The CNTFRQ characteristics are:

Purpose

This register is provided so that software can discover the frequency of the system counter. It must be programmed with this value as part of system initialization. The value of the register is not interpreted by hardware.

Configuration

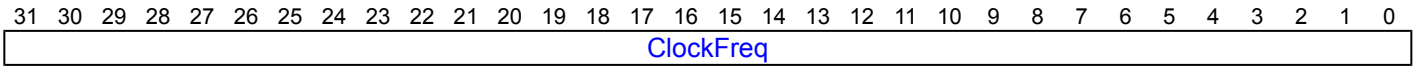
AArch32 System register CNTFRQ bits [31:0] are architecturally mapped to AArch64 System register CNTFRQ_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTFRQ are UNDEFINED.

Attributes

CNTFRQ is a 32-bit register.

Field descriptions



ClockFreq, bits [31:0]

Clock frequency. Indicates the system counter clock frequency, in Hz.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFRQ

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.<EL0PCTEN,EL0VCTEN> == '00' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PCTEN == '0' &&
CNTKCTL.PL0VCTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.<EL0PCTEN,EL0VCTEN> == '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        R[t] = CNTFRQ;
elsif PSTATE.EL == EL1 then
    R[t] = CNTFRQ;
elsif PSTATE.EL == EL2 then
    R[t] = CNTFRQ;
elsif PSTATE.EL == EL3 then
    R[t] = CNTFRQ;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    CNTFRQ = R[t];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHCTL, Counter-timer Hyp Control register

The CNTHCTL characteristics are:

Purpose

Controls the generation of an event stream from the physical counter, and access from Non-secure EL1 modes to the physical counter and the Non-secure EL1 physical timer.

Configuration

AArch32 System register CNTHCTL bits [31:0] are architecturally mapped to AArch64 System register CNTHCTL_EL2[31:0].

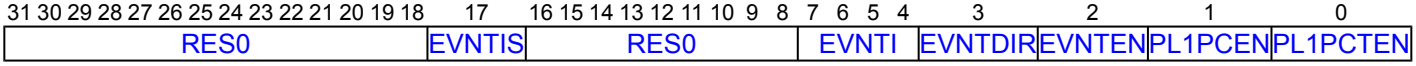
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to CNTHCTL are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHCTL is a 32-bit register.

Field descriptions



Bits [31:18]

Reserved, RES0.

EVNTIS, bit [17]

When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

EVNTIS	Meaning
0b0	The CNTHCTL.EVNTI field applies to CNTPCT[15:0].
0b1	The CNTHCTL.EVNTI field applies to CNTPCT[23:8].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:8]

Reserved, RES0.

EVNTI, bits [7:4]

Selects which bit of CNTPCT, as seen from EL2, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTHCTL.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of CNTPCT.

Otherwise, this field selects a trigger bit in the range 0 to 15 of CNTPCT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the CNTPCT trigger bit, as seen from EL2 and defined by EVNTI, generates an event when the event stream is enabled.

EVNTDIR	Meaning
0b0	A 0 to 1 transition of the trigger bit triggers an event.
0b1	A 1 to 0 transition of the trigger bit triggers an event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

Enables the generation of an event stream from CNTPCT as seen from EL2.

EVNTEN	Meaning
0b0	Disables the event stream.
0b1	Enables the event stream.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PL1PCEN, bit [1]

Traps Non-secure EL0 and EL1 MRC or MCR accesses, reported using EC syndrome value 0x03, and MRRC or MCRR accesses, reported using EC syndrome value 0x04, to the physical timer registers to Hyp mode.

PL1PCEN	Meaning
0b0	Non-secure EL0 and EL1 accesses to the CNTP_CTL, CNTP_CVAL, and CNTP_TVAL are trapped to Hyp mode, unless trapped by CNTKCTL.PL0PTEN.
0b1	This control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PL1PCTEN, bit [0]

Traps Non-secure EL0 and EL1 MRRC or MCRR accesses, reported using EC syndrome value 0x04, to the physical counter register to Hyp mode.

PL1PCTEN	Meaning
0b0	Non-secure EL0 and EL1 accesses to the CNTPCT are trapped to Hyp mode, unless it is trapped by CNTKCTL.PL0PCTEN.
0b1	This control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHCTL

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = CNTHCTL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = CNTHCTL;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHCTL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTHCTL = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHP_CTL, Counter-timer Hyp Physical Timer Control register

The CNTHP_CTL characteristics are:

Purpose

Control register for the Hyp mode physical timer.

Configuration

This register is banked between CNTHP_CTL and CNTHP_CTL_S and CNTHP_CTL_NS.

AArch32 System register CNTHP_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTHP_CTL_EL2[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTHP_CTL are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

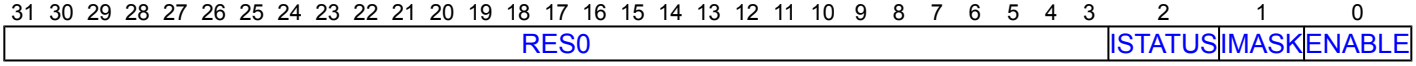
Attributes

CNTHP_CTL is a 32-bit register.

This register has the following instances:

- CNTHP_CTL, when EL3 is not implemented or FEAT_AA64 is implemented.
- CNTHP_CTL_S, when FEAT_AA32EL3 is implemented.
- CNTHP_CTL_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHP_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_CTL

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = CNTHP_CTL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = CNTHP_CTL;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b100	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CTL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTHP_CTL = R[t];
```

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t] = CNTHPS_CTL_EL2<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t] = CNTHP_CTL_EL2<31:0>;
    else
        R[t] = CNTP_CTL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = CNTP_CTL_NS;
    else
        R[t] = CNTP_CTL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = CNTP_CTL_NS;
    else
        R[t] = CNTP_CTL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = CNTP_CTL_S;
    else
        R[t] = CNTP_CTL_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CTL_EL2 = R[t];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CTL_S = R[t];
    else
        CNTP_CTL_NS = R[t];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHP_CVAL, Counter-timer Hyp Physical CompareValue register

The CNTHP_CVAL characteristics are:

Purpose

Holds the compare value for the Hyp mode physical timer.

Configuration

This register is banked between CNTHP_CVAL and CNTHP_CVAL_S and CNTHP_CVAL_NS.

AArch32 System register CNTHP_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTHP_CVAL_EL2[63:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTHP_CVAL are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

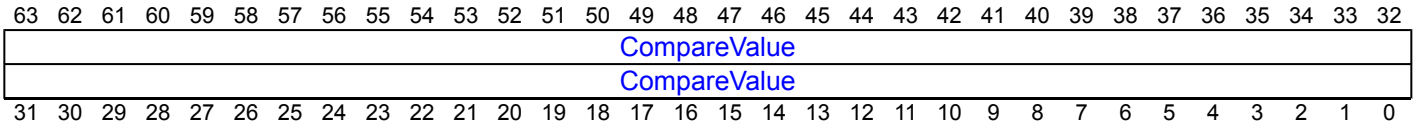
Attributes

CNTHP_CVAL is a 64-bit register.

This register has the following instances:

- CNTHP_CVAL, when EL3 is not implemented or FEAT_AA64 is implemented.
- CNTHP_CVAL_S, when FEAT_AA32EL3 is implemented.
- CNTHP_CVAL_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When CNTHP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHP_CTL.ISTATUS is set to 1.
- If CNTHP_CTL.IMASK is 0, an interrupt is generated.

When CNTHP_CTL.ENABLE is 0, the timer condition is not met, but CNTPCT continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_CVAL

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0110

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t, t2] = CNTHP_CVAL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t, t2] = CNTHP_CVAL;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0110

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL = R[t, t2];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTHP_CVAL = R[t, t2];
```

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0010

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t, t2] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t, t2] = CNTHP_CVAL_EL2;
    else
        R[t, t2] = CNTP_CVAL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = CNTP_CVAL_NS;
    else
        R[t, t2] = CNTP_CVAL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = CNTP_CVAL_NS;
    else
        R[t, t2] = CNTP_CVAL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t, t2] = CNTP_CVAL_S;
    else
        R[t, t2] = CNTP_CVAL_NS;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0010

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = R[t, t2];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = R[t, t2];
    else
        CNTP_CVAL = R[t, t2];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t, t2];
    else
        CNTP_CVAL = R[t, t2];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t, t2];
    else
        CNTP_CVAL = R[t, t2];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = R[t, t2];
    else
        CNTP_CVAL_NS = R[t, t2];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHP_TVAL, Counter-timer Hyp Physical Timer TimerValue register

The CNTHP_TVAL characteristics are:

Purpose

Holds the timer value for the Hyp mode physical timer.

Configuration

This register is banked between CNTHP_TVAL and CNTHP_TVAL_S and CNTHP_TVAL_NS.

AArch32 System register CNTHP_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTHP_TVAL_EL2[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTHP_TVAL are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

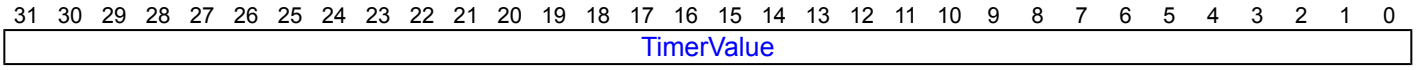
Attributes

CNTHP_TVAL is a 32-bit register.

This register has the following instances:

- CNTHP_TVAL, when EL3 is not implemented or FEAT_AA64 is implemented.
- CNTHP_TVAL_S, when FEAT_AA32EL3 is implemented.
- CNTHP_TVAL_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If CNTHP_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHP_CTL.ENABLE is 1, the value returned is (CNTHP_CVAL - CNTPCT).

On a write of this register, CNTHP_CVAL is set to (CNTPCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CNTHP_CVAL) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHP_CTL.ISTATUS is set to 1.
- If CNTHP_CTL.IMASK is 0, an interrupt is generated.

When CNTHP_CTL.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_TVAL

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if CNTHP_CTL.ENABLE == '0' then
        R[t] = bits(32) UNKNOWN;
    else
        R[t] = (CNTHP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        if CNTHP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHP_CVAL - PhysicalCountInt())<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTHP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
```

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
(IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            if CNTP_CTL_NS.ENABLE == '0' then
                R[t] = bits(32) UNKNOWN;
            else
                R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
        else
            if CNTP_CTL_S.ENABLE == '0' then
                R[t] = bits(32) UNKNOWN;
            else
                R[t] = (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
```

```
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if CNTP_CTL_NS.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if CNTP_CTL_NS.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        if CNTP_CTL_S.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL_NS.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
(IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        CNTP_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
        else
            CNTP_CVAL_S = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
```

```
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHPS_CTL, Counter-timer Secure Physical Timer Control Register (EL2)

The CNTHPS_CTL characteristics are:

Purpose

Provides AArch32 access from EL0 to the Secure EL2 physical timer.

Configuration

This register is banked between CNTHPS_CTL and CNTHPS_CTL_S and CNTHPS_CTL_NS.

AArch32 System register CNTHPS_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTHPS_CTL_EL2[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHPS_CTL are UNDEFINED.

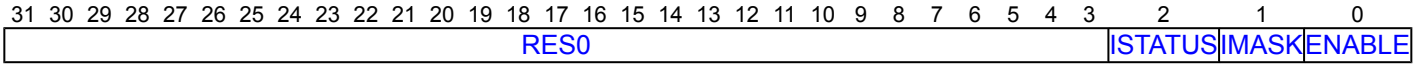
Attributes

CNTHPS_CTL is a 32-bit register.

This register has the following instances:

- CNTHPS_CTL, when EL3 is not implemented or FEAT_AA64 is implemented.
- CNTHPS_CTL_S, when FEAT_AA32EL3 is implemented.
- CNTHPS_CTL_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the CNTHPS_CTL.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the CNTHPS_CTL.ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHPS_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_CTL

This register is accessed using the encoding for CNTP_CTL.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t] = CNTHPS_CTL_EL2<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t] = CNTHP_CTL_EL2<31:0>;
    else
        R[t] = CNTP_CTL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = CNTP_CTL_NS;
    else
        R[t] = CNTP_CTL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = CNTP_CTL_NS;
    else
        R[t] = CNTP_CTL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = CNTP_CTL_S;
    else
        R[t] = CNTP_CTL_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CTL_EL2 = R[t];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CTL_S = R[t];
    else
        CNTP_CTL_NS = R[t];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHPS_CVAL, Counter-timer Secure Physical Timer CompareValue Register (EL2)

The CNTHPS_CVAL characteristics are:

Purpose

Provides AArch32 access from EL0 to the compare value for the Secure EL2 physical timer.

Configuration

This register is banked between CNTHPS_CVAL and CNTHPS_CVAL_S and CNTHPS_CVAL_NS.

AArch32 System register CNTHPS_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTHPS_CVAL_EL2[63:0].

This register is present only when FEAT_AA32 is implemented and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHPS_CVAL are UNDEFINED.

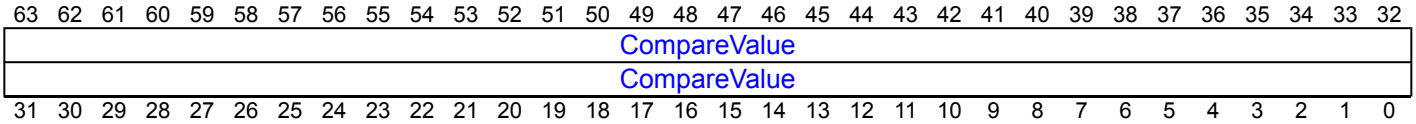
Attributes

CNTHPS_CVAL is a 64-bit register.

This register has the following instances:

- CNTHPS_CVAL, when EL3 is not implemented or FEAT_AA64 is implemented.
- CNTHPS_CVAL_S, when FEAT_AA32EL3 is implemented.
- CNTHPS_CVAL_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When CNTHPS_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHPS_CTL.ISTATUS is set to 1.
- If CNTHPS_CTL.IMASK is 0, an interrupt is generated.

When CNTHPS_CTL.ENABLE is 0, the timer condition is not met, but CNTPCT continues to count

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_CVAL

This register is accessed using the encoding for CNTP_CVAL.

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0010

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t, t2] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t, t2] = CNTHP_CVAL_EL2;
    else
        R[t, t2] = CNTP_CVAL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = CNTP_CVAL_NS;
    else
        R[t, t2] = CNTP_CVAL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = CNTP_CVAL_NS;
    else
        R[t, t2] = CNTP_CVAL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t, t2] = CNTP_CVAL_S;
    else
        R[t, t2] = CNTP_CVAL_NS;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0010

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = R[t, t2];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = R[t, t2];
    else
        CNTP_CVAL = R[t, t2];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t, t2];
    else
        CNTP_CVAL = R[t, t2];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t, t2];
    else
        CNTP_CVAL = R[t, t2];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = R[t, t2];
    else
        CNTP_CVAL_NS = R[t, t2];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHPS_TVAL, Counter-timer Secure Physical Timer TimerValue Register (EL2)

The CNTHPS_TVAL characteristics are:

Purpose

Provides AArch32 access from EL0 to the timer value for the Secure EL2 physical timer.

Configuration

This register is banked between CNTHPS_TVAL and CNTHPS_TVAL_S and CNTHPS_TVAL_NS.

AArch32 System register CNTHPS_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTHPS_TVAL_EL2[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHPS_TVAL are UNDEFINED.

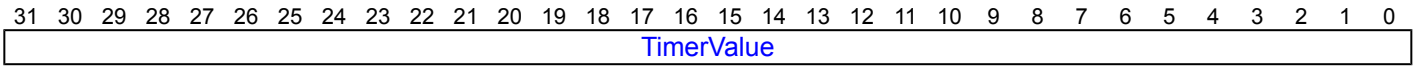
Attributes

CNTHPS_TVAL is a 32-bit register.

This register has the following instances:

- CNTHPS_TVAL, when EL3 is not implemented or FEAT_AA64 is implemented.
- CNTHPS_TVAL_S, when FEAT_AA32EL3 is implemented.
- CNTHPS_TVAL_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If CNTHPS_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHPS_CTL.ENABLE is 1, the value returned is (CNTHPS_CVAL - CNTPCT).

On a write of this register, CNTHPS_CVAL is set to (CNTPCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHPS_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CNTHPS_CVAL) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHPS_CTL.ISTATUS is set to 1.
- If CNTHPS_CTL.IMASK is 0, an interrupt is generated.

When CNTHPS_CTL.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_TVAL

This register is accessed using the encoding for CNTP_TVAL.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
(IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            if CNTP_CTL_NS.ENABLE == '0' then
                R[t] = bits(32) UNKNOWN;
            else
                R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
        else
            if CNTP_CTL_S.ENABLE == '0' then
                R[t] = bits(32) UNKNOWN;
            else
                R[t] = (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
```

```
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if CNTP_CTL_NS.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if CNTP_CTL_NS.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        if CNTP_CTL_S.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL_NS.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
(IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        CNTP_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
        else
            CNTP_CVAL_S = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
```

```
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_CTL, Counter-timer Virtual Timer Control register (EL2)

The CNTHV_CTL characteristics are:

Purpose

Provides AArch32 access to the control register for the EL2 virtual timer.

Configuration

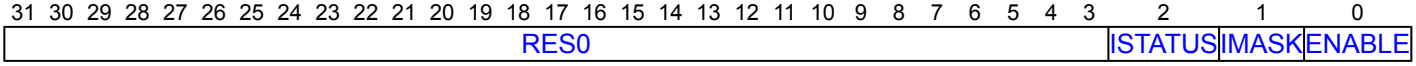
AArch32 System register CNTHV_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTHV_CTL_EL2[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_VHE is implemented. Otherwise, direct accesses to CNTHV_CTL are UNDEFINED.

Attributes

CNTHV_CTL is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHV_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHV_CTL

This register is accessed using the encoding for CNTV_CTL.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t] = CNTHVS_CTL_EL2<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t] = CNTHV_CTL_EL2<31:0>;
    else
        R[t] = CNTV_CTL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        R[t] = CNTV_CTL;
elsif PSTATE.EL == EL2 then
    R[t] = CNTV_CTL;
elsif PSTATE.EL == EL3 then
    R[t] = CNTV_CTL;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CTL_EL2 = R[t];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = R[t];
    else
        CNTV_CTL = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        CNTV_CTL = R[t];
elsif PSTATE.EL == EL2 then
    CNTV_CTL = R[t];
elsif PSTATE.EL == EL3 then
    CNTV_CTL = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_CVAL, Counter-timer Virtual Timer CompareValue register (EL2)

The CNTHV_CVAL characteristics are:

Purpose

Provides AArch32 access to the compare value for the EL2 virtual timer.

Configuration

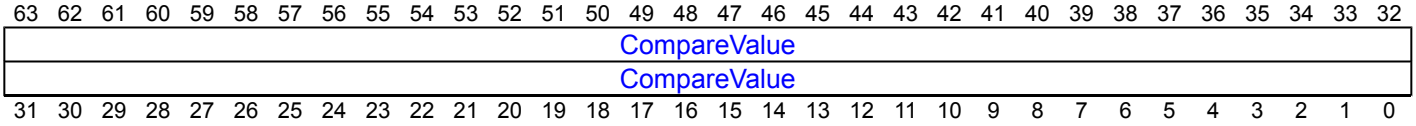
AArch32 System register CNTHV_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTHV_CVAL_EL2[63:0].

This register is present only when FEAT_AA32 is implemented and FEAT_VHE is implemented. Otherwise, direct accesses to CNTHV_CVAL are UNDEFINED.

Attributes

CNTHV_CVAL is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the EL2 virtual timer CompareValue.

When CNTHV_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHV_CTL.ISTATUS is set to 1.
- If CNTHV_CTL.IMASK is 0, an interrupt is generated.

When CNTHV_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

Accessing CNTHV_CVAL

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0011

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t, t2] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t, t2] = CNTHV_CVAL_EL2;
    else
        R[t, t2] = CNTV_CVAL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        R[t, t2] = CNTV_CVAL;
elsif PSTATE.EL == EL2 then
    R[t, t2] = CNTV_CVAL;
elsif PSTATE.EL == EL3 then
    R[t, t2] = CNTV_CVAL;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0011

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = R[t, t2];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = R[t, t2];
    else
        CNTV_CVAL = R[t, t2];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        CNTV_CVAL = R[t, t2];
elsif PSTATE.EL == EL2 then
    CNTV_CVAL = R[t, t2];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_TVAL, Counter-timer Virtual Timer TimerValue register (EL2)

The CNTHV_TVAL characteristics are:

Purpose

Provides AArch32 access to the timer value for the EL2 virtual timer.

Configuration

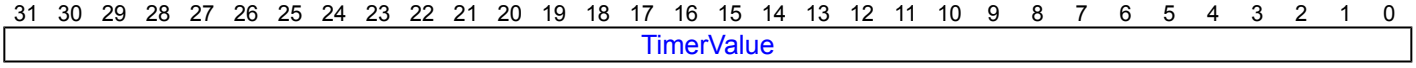
AArch32 System register CNTHV_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTHV_TVAL_EL2[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_VHE is implemented. Otherwise, direct accesses to CNTHV_TVAL are UNDEFINED.

Attributes

CNTHV_TVAL is a 32-bit register.

Field descriptions



TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If CNTHV_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHV_CTL.ENABLE is 1, the value returned is (CNTHV_CVAL - CNTVCT).

On a write of this register, CNTHV_CVAL is set to (CNTVCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHV_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CNTHV_CVAL) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHV_CTL.ISTATUS is set to 1.
- If CNTHV_CTL.IMASK is 0, an interrupt is generated.

When CNTHV_CTL.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

Accessing CNTHV_TVAL

This register is accessed using the encoding for CNTV_TVAL.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>;
    else
        if CNTV_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            R[t] = (CNTV_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        if CNTV_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            R[t] = (CNTV_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL2 then
    if CNTV_CTL.ENABLE == '0' then
        R[t] = bits(32) UNKNOWN;
    else
        R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
elsif PSTATE.EL == EL3 then
    if CNTV_CTL.ENABLE == '0' then
        R[t] = bits(32) UNKNOWN;
    elsif HaveEL(EL2) then
        R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
    else
```

```
        R[t] = (CNTV_CVAL - PhysicalCountInt())<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) then
        CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
    else
        CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHVS_CTL, Counter-timer Secure Virtual Timer Control Register (EL2)

The CNTHVS_CTL characteristics are:

Purpose

Provides AArch32 access from EL0 to the Secure EL2 virtual timer.

Configuration

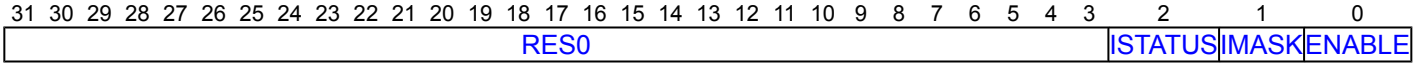
AArch32 System register CNTHVS_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTHVS_CTL_EL2[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHVS_CTL are UNDEFINED.

Attributes

CNTHVS_CTL is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHVS_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_CTL

This register is accessed using the encoding for CNTV_CTL.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t] = CNTHVS_CTL_EL2<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t] = CNTHV_CTL_EL2<31:0>;
    else
        R[t] = CNTV_CTL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        R[t] = CNTV_CTL;
elsif PSTATE.EL == EL2 then
    R[t] = CNTV_CTL;
elsif PSTATE.EL == EL3 then
    R[t] = CNTV_CTL;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CTL_EL2 = R[t];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = R[t];
    else
        CNTV_CTL = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        CNTV_CTL = R[t];
elsif PSTATE.EL == EL2 then
    CNTV_CTL = R[t];
elsif PSTATE.EL == EL3 then
    CNTV_CTL = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHVS_CVAL, Counter-timer Secure Virtual Timer CompareValue Register (EL2)

The CNTHVS_CVAL characteristics are:

Purpose

Provides AArch32 access to the compare value for the Secure EL2 virtual timer.

Configuration

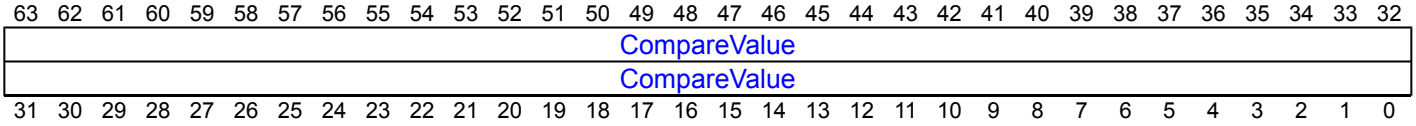
AArch32 System register CNTHVS_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTHVS_CVAL_EL2[63:0].

This register is present only when FEAT_AA32 is implemented and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHVS_CVAL are UNDEFINED.

Attributes

CNTHVS_CVAL is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the EL2 virtual timer CompareValue.

When CNTHVS_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHVS_CTL.ISTATUS is set to 1.
- If CNTHVS_CTL.IMASK is 0, an interrupt is generated.

When CNTHVS_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

Accessing CNTHVS_CVAL

This register is accessed using the encoding for CNTV_CVAL.

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0011

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t, t2] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t, t2] = CNTHV_CVAL_EL2;
    else
        R[t, t2] = CNTV_CVAL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        R[t, t2] = CNTV_CVAL;
elsif PSTATE.EL == EL2 then
    R[t, t2] = CNTV_CVAL;
elsif PSTATE.EL == EL3 then
    R[t, t2] = CNTV_CVAL;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0011

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = R[t, t2];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = R[t, t2];
    else
        CNTV_CVAL = R[t, t2];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        CNTV_CVAL = R[t, t2];
elsif PSTATE.EL == EL2 then
    CNTV_CVAL = R[t, t2];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHVS_TVAL, Counter-timer Secure Virtual Timer TimerValue Register (EL2)

The CNTHVS_TVAL characteristics are:

Purpose

Provides AArch32 access to the timer value for the Secure EL2 virtual timer.

Configuration

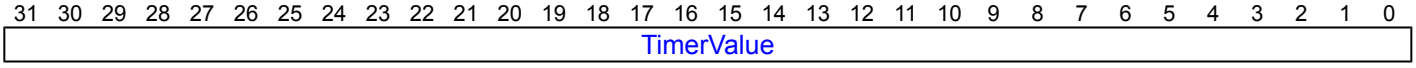
AArch32 System register CNTHVS_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTHVS_TVAL_EL2[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHVS_TVAL are UNDEFINED.

Attributes

CNTHVS_TVAL is a 32-bit register.

Field descriptions



TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If CNTHVS_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHVS_CTL.ENABLE is 1, the value returned is (CNTHVS_CVAL - CNTVCT).

On a write of this register, CNTHVS_CVAL is set to (CNTVCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHVS_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CNTHVS_CVAL) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHVS_CTL.ISTATUS is set to 1.
- If CNTHVS_CTL.IMASK is 0, an interrupt is generated.

When CNTHVS_CTL.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

Accessing CNTHVS_TVAL

This register is accessed using the encoding for CNTV_TVAL.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>;
    else
        if CNTV_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            R[t] = (CNTV_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        if CNTV_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            R[t] = (CNTV_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL2 then
    if CNTV_CTL.ENABLE == '0' then
        R[t] = bits(32) UNKNOWN;
    else
        R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
elsif PSTATE.EL == EL3 then
    if CNTV_CTL.ENABLE == '0' then
        R[t] = bits(32) UNKNOWN;
    elsif HaveEL(EL2) then
        R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
    else
```

```
        R[t] = (CNTV_CVAL - PhysicalCountInt())<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) then
        CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
    else
        CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTKCTL, Counter-timer Kernel Control register

The CNTKCTL characteristics are:

Purpose

Controls the generation of an event stream from the virtual counter, and access from EL0 modes to the physical counter, virtual counter, EL1 physical timers, and the virtual timer.

Configuration

AArch32 System register CNTKCTL bits [31:0] are architecturally mapped to AArch64 System register CNTKCTL_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to CNTKCTL are UNDEFINED.

Attributes

CNTKCTL is a 32-bit register.

Field descriptions

Bits [31:18]

Reserved, RES0.

EVNTIS, bit [17]

When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

EVNTIS	Meaning
0b0	The CNTKCTL.EVNTI field applies to CNTVCT[15:0].
0b1	The CNTKCTL.EVNTI field applies to CNTVCT[23:8].

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:10]

Reserved, RES0.

PL0PTEN, bit [9]

Traps PL0 accesses to the physical timer registers to Undefined mode.

PL0PTEN	Meaning
0b0	PL0 accesses to the CNTP_CTL, CNTP_CVAL, and CNTP_TVAL registers are trapped to Undefined mode.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PL0VTEN, bit [8]

Traps PL0 accesses to the virtual timer registers to Undefined mode.

PL0VTEN	Meaning
0b0	PL0 accesses to the CNTV_CTL, CNTV_CVAL, and CNTV_TVAL registers are trapped to Undefined mode.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTI, bits [7:4]

Selects which bit of CNTVCT, as seen from EL1, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTKCTL.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of CNTVCT.

Otherwise, this field selects a trigger bit in the range 0 to 15 of CNTVCT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the CNTVCT trigger bit, as seen from EL1 and defined by EVNTI, generates an event when the event stream is enabled.

EVNTDIR	Meaning
0b0	A 0 to 1 transition of the trigger bit triggers an event.
0b1	A 1 to 0 transition of the trigger bit triggers an event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

Enables the generation of an event stream from CNTVCT as seen from EL1.

EVNTEN	Meaning
0b0	Disables the event stream.
0b1	Enables the event stream.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PL0VCTEN, bit [1]

Traps PL0 accesses to the frequency register and virtual counter register to Undefined mode.

PL0VCTEN	Meaning
0b0	PL0 accesses to the CNTVCT are trapped to Undefined mode. PL0 accesses to the CNTFRQ register are trapped to Undefined mode, if CNTKCTL.PL0PCTEN is also 0.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PL0PCTEN, bit [0]

Traps PL0 accesses to the frequency register and physical counter register to Undefined mode.

PL0PCTEN	Meaning
0b0	PL0 accesses to the CNTPCT are trapped to Undefined mode. PL0 accesses to the CNTFRQ register are trapped to Undefined mode, if CNTKCTL.PL0VCTEN is also 0.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTKCTL

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    R[t] = CNTKCTL;
elsif PSTATE.EL == EL2 then
    R[t] = CNTKCTL;
elsif PSTATE.EL == EL3 then
    R[t] = CNTKCTL;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL = R[t];
elsif PSTATE.EL == EL2 then
    CNTKCTL = R[t];
elsif PSTATE.EL == EL3 then
    CNTKCTL = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_CTL, Counter-timer Physical Timer Control register

The CNTP_CTL characteristics are:

Purpose

Control register for the EL1 physical timer.

Configuration

This register is banked between CNTP_CTL and CNTP_CTL_S and CNTP_CTL_NS.

AArch32 System register CNTP_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTP_CTL_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTP_CTL are UNDEFINED.

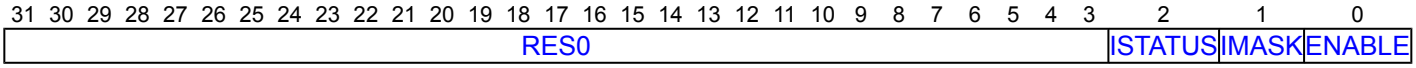
Attributes

CNTP_CTL is a 32-bit register.

This register has the following instances:

- CNTP_CTL, when EL3 is not implemented or FEAT_AA64 is implemented.
- CNTP_CTL_S, when FEAT_AA32EL3 is implemented.
- CNTP_CTL_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTP_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing CNTP_CTL

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t] = CNTHPS_CTL_EL2<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t] = CNTHP_CTL_EL2<31:0>;
    else
        R[t] = CNTP_CTL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = CNTP_CTL_NS;
    else
        R[t] = CNTP_CTL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = CNTP_CTL_NS;
    else
        R[t] = CNTP_CTL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = CNTP_CTL_S;
    else
        R[t] = CNTP_CTL_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CTL_EL2 = R[t];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CTL_S = R[t];
    else
        CNTP_CTL_NS = R[t];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_CVAL, Counter-timer Physical Timer CompareValue register

The CNTP_CVAL characteristics are:

Purpose

Holds the compare value for the EL1 physical timer.

Configuration

This register is banked between CNTP_CVAL and CNTP_CVAL_S and CNTP_CVAL_NS.

AArch32 System register CNTP_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTP_CVAL_EL0[63:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTP_CVAL are UNDEFINED.

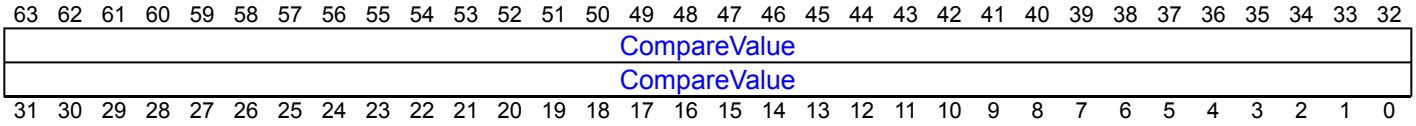
Attributes

CNTP_CVAL is a 64-bit register.

This register has the following instances:

- CNTP_CVAL, when EL3 is not implemented or FEAT_AA64 is implemented.
- CNTP_CVAL_S, when FEAT_AA32EL3 is implemented.
- CNTP_CVAL_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



CompareValue, bits [63:0]

Holds the EL1 physical timer CompareValue.

When CNTP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTP_CTL.ISTATUS is set to 1.
- If CNTP_CTL.IMASK is 0, an interrupt is generated.

When CNTP_CTL.ENABLE is 0, the timer condition is not met, but CNTPCT continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CVAL

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0010

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t, t2] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t, t2] = CNTHP_CVAL_EL2;
    else
        R[t, t2] = CNTP_CVAL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = CNTP_CVAL_NS;
    else
        R[t, t2] = CNTP_CVAL;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = CNTP_CVAL_NS;
    else
        R[t, t2] = CNTP_CVAL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t, t2] = CNTP_CVAL_S;
    else
        R[t, t2] = CNTP_CVAL_NS;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0010

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = R[t, t2];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = R[t, t2];
    else
        CNTP_CVAL = R[t, t2];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t, t2];
    else
        CNTP_CVAL = R[t, t2];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t, t2];
    else
        CNTP_CVAL = R[t, t2];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = R[t, t2];
    else
        CNTP_CVAL_NS = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_TVAL, Counter-timer Physical Timer TimerValue register

The CNTP_TVAL characteristics are:

Purpose

Holds the timer value for the EL1 physical timer.

Configuration

This register is banked between CNTP_TVAL and CNTP_TVAL_S and CNTP_TVAL_NS.

AArch32 System register CNTP_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTP_TVAL_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTP_TVAL are UNDEFINED.

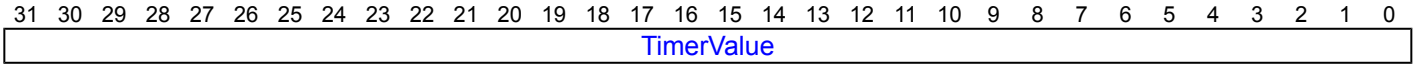
Attributes

CNTP_TVAL is a 32-bit register.

This register has the following instances:

- CNTP_TVAL, when EL3 is not implemented or FEAT_AA64 is implemented.
- CNTP_TVAL_S, when FEAT_AA32EL3 is implemented.
- CNTP_TVAL_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



TimerValue, bits [31:0]

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If CNTP_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTP_CTL.ENABLE is 1, the value returned is (CNTP_CVAL - CNTPCT).

On a write of this register, CNTP_CVAL is set to (CNTPCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CNTP_CVAL) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTP_CTL.ISTATUS is set to 1.
- If CNTP_CTL.IMASK is 0, an interrupt is generated.

When CNTP_CTL.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_TVAL

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
(IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            if CNTP_CTL_NS.ENABLE == '0' then
                R[t] = bits(32) UNKNOWN;
            else
                R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
        else
            if CNTP_CTL_S.ENABLE == '0' then
                R[t] = bits(32) UNKNOWN;
            else
                R[t] = (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
```

```
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if CNTP_CTL_NS.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if CNTP_CTL_NS.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        if CNTP_CTL_S.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL_NS.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHPS_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
(IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        CNTP_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
        else
            CNTP_CVAL_S = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
```

```
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        CNTP_CVAL_NS = SignExtend(R[t], 64) + PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPCT, Counter-timer Physical Count register

The CNTPCT characteristics are:

Purpose

Holds the 64-bit physical count value.

Configuration

AArch32 System register CNTPCT bits [63:0] are architecturally mapped to AArch64 System register CNTPCT_EL0[63:0].

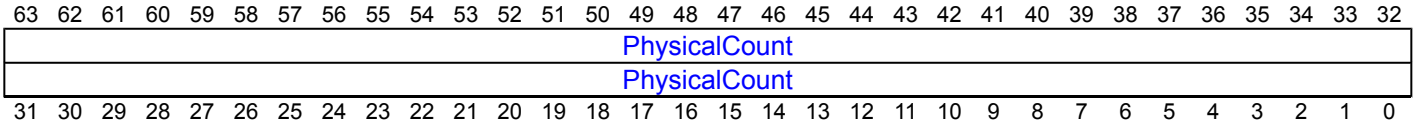
This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTPCT are UNDEFINED.

All reads to the CNTPCT occur in program order relative to reads to CNTPCTSS or CNTPCT.

Attributes

CNTPCT is a 64-bit register.

Field descriptions



PhysicalCount, bits [63:0]

Physical count value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCT

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PCTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCTEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
            R[t, t2] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCTEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
            R[t, t2] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    R[t, t2] = PhysicalCountInt();
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPCTSS, Counter-timer Self-Synchronized Physical Count register

The CNTPCTSS characteristics are:

Purpose

Holds the 64-bit physical count value.

Configuration

AArch32 System register CNTPCTSS bits [63:0] are architecturally mapped to AArch64 System register CNTPCTSS_EL0[63:0].

This register is present only when FEAT_AA32 is implemented and FEAT_ECV is implemented. Otherwise, direct accesses to CNTPCTSS are UNDEFINED.

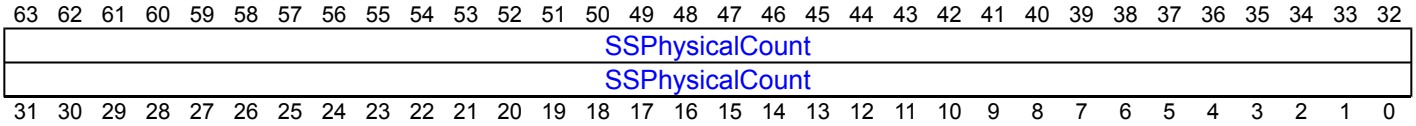
All reads to the CNTPCTSS occur in program order relative to reads to CNTPCT or CNTPCTSS.

This register is a view of the CNTPCT register for which reads appear to occur in program order relative to other instructions, without the need for any explicit synchronization. Reads of this register return a value consistent with the counter not being read until the read instruction is known to be non-speculative.

Attributes

CNTPCTSS is a 64-bit register.

Field descriptions



SSPhysicalCount, bits [63:0]

Self-Synchronized Physical count value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCTSS

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b1000

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_ECV)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0PCTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL2) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
CNTHCTL.PL1PCTEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
            R[t, t2] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
CNTHCTL.PL1PCTEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() &&
IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
'1') && CNTHCTL_EL2.ECV == '1' then
            R[t, t2] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    R[t, t2] = PhysicalCountInt();
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_CTL, Counter-timer Virtual Timer Control register

The CNTV_CTL characteristics are:

Purpose

Control register for the virtual timer.

Configuration

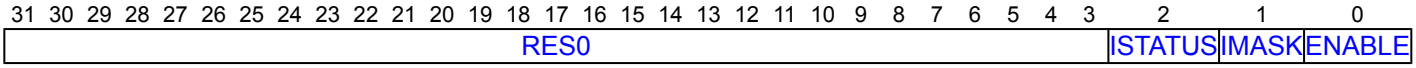
AArch32 System register CNTV_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTV_CTL_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTV_CTL are UNDEFINED.

Attributes

CNTV_CTL is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTV_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing CNTV_CTL

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t] = CNTHVS_CTL_EL2<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t] = CNTHV_CTL_EL2<31:0>;
    else
        R[t] = CNTV_CTL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        R[t] = CNTV_CTL;
elsif PSTATE.EL == EL2 then
    R[t] = CNTV_CTL;
elsif PSTATE.EL == EL3 then
    R[t] = CNTV_CTL;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CTL_EL2 = R[t];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = R[t];
    else
        CNTV_CTL = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        CNTV_CTL = R[t];
elsif PSTATE.EL == EL2 then
    CNTV_CTL = R[t];
elsif PSTATE.EL == EL3 then
    CNTV_CTL = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_CVAL, Counter-timer Virtual Timer CompareValue register

The CNTV_CVAL characteristics are:

Purpose

Holds the compare value for the virtual timer.

Configuration

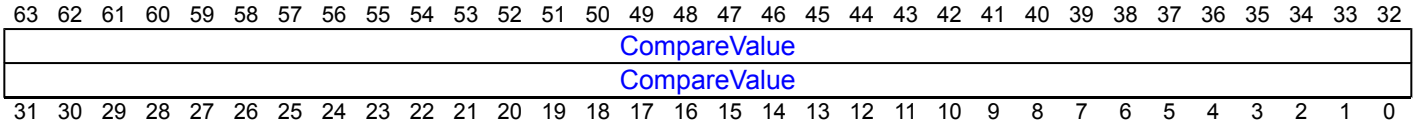
AArch32 System register CNTV_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTV_CVAL_EL0[63:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTV_CVAL are UNDEFINED.

Attributes

CNTV_CVAL is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the EL1 virtual timer CompareValue.

When CNTV_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTV_CTL.ISTATUS is set to 1.
- If CNTV_CTL.IMASK is 0, an interrupt is generated.

When CNTV_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CVAL

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0011

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        R[t, t2] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        R[t, t2] = CNTHV_CVAL_EL2;
    else
        R[t, t2] = CNTV_CVAL;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        R[t, t2] = CNTV_CVAL;
elsif PSTATE.EL == EL2 then
    R[t, t2] = CNTV_CVAL;
elsif PSTATE.EL == EL3 then
    R[t, t2] = CNTV_CVAL;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0011

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = R[t, t2];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = R[t, t2];
    else
        CNTV_CVAL = R[t, t2];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        CNTV_CVAL = R[t, t2];
elsif PSTATE.EL == EL2 then
    CNTV_CVAL = R[t, t2];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_TVAL, Counter-timer Virtual Timer TimerValue register

The CNTV_TVAL characteristics are:

Purpose

Holds the timer value for the virtual timer.

Configuration

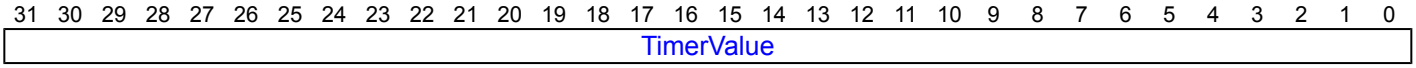
AArch32 System register CNTV_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTV_TVAL_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTV_TVAL are UNDEFINED.

Attributes

CNTV_TVAL is a 32-bit register.

Field descriptions



TimerValue, bits [31:0]

The TimerValue view of the virtual timer.

On a read of this register:

- If CNTV_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTV_CTL.ENABLE is 1, the value returned is (CNTV_CVAL - CNTVCT).

On a write of this register, CNTV_CVAL is set to (CNTVCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTP_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CNTP_CVAL) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTV_CTL.ISTATUS is set to 1.
- If CNTV_CTL.IMASK is 0, an interrupt is generated.

When CNTV_CTL.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_TVAL

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        else
            R[t] = (CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>;
    else
        if CNTV_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            R[t] = (CNTV_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        if CNTV_CTL.ENABLE == '0' then
            R[t] = bits(32) UNKNOWN;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            R[t] = (CNTV_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL2 then
    if CNTV_CTL.ENABLE == '0' then
        R[t] = bits(32) UNKNOWN;
    else
        R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
elsif PSTATE.EL == EL3 then
    if CNTV_CTL.ENABLE == '0' then
        R[t] = bits(32) UNKNOWN;
    elsif HaveEL(EL2) then
        R[t] = (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
    else
```

```
        R[t] = (CNTV_CVAL - PhysicalCountInt())<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
then
        CNTHVS_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(R[t], 64) + PhysicalCountInt();
    else
        if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) then
        CNTV_CVAL = (SignExtend(R[t], 64) + PhysicalCountInt()) - CNTVOFF;
    else
        CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVCT, Counter-timer Virtual Count register

The CNTVCT characteristics are:

Purpose

Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value minus the virtual offset visible in CNTVOFF.

Configuration

AArch32 System register CNTVCT bits [63:0] are architecturally mapped to AArch64 System register CNTVCT_EL0[63:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CNTVCT are UNDEFINED.

The value of this register is the same as the value of CNTPCT in the following conditions:

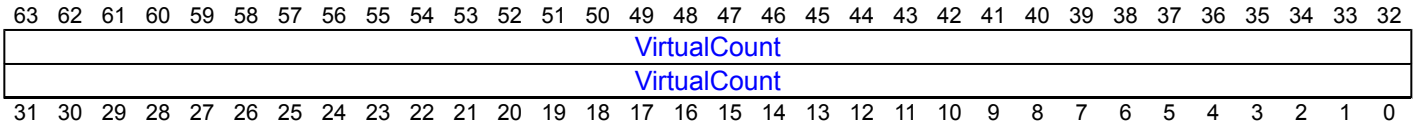
- When EL2 is not implemented.
- When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, and this register is read from Non-secure EL0.

All reads to the CNTVCT occur in program order relative to reads to CNTVCTSS or CNTVCT.

Attributes

CNTVCT is a 64-bit register.

Field descriptions



VirtualCount, bits [63:0]

Virtual count value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCT

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VCTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
(!EL2Enabled() || !ELIsInHost(EL0)) then
            R[t, t2] = PhysicalCountInt() - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) && !ELUsingAArch32(EL2) &&
(!EL2Enabled() || !ELIsInHost(EL0)) then
            R[t, t2] = PhysicalCountInt() - CNTVOFF;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            R[t, t2] = PhysicalCountInt() - CNTVOFF;
        else
            R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            R[t, t2] = PhysicalCountInt() - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            R[t, t2] = PhysicalCountInt() - CNTVOFF;
        else
            R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL2) then
        R[t, t2] = PhysicalCountInt() - CNTVOFF;
    else
        R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        R[t, t2] = PhysicalCountInt() - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        R[t, t2] = PhysicalCountInt() - CNTVOFF;
    else
        R[t, t2] = PhysicalCountInt();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVCTSS, Counter-timer Self-Synchronized Virtual Count register

The CNTVCTSS characteristics are:

Purpose

Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value visible in CNTPCT minus the virtual offset visible in CNTVOFF.

Configuration

AArch32 System register CNTVCTSS bits [63:0] are architecturally mapped to AArch64 System register CNTVCTSS_EL0[63:0].

This register is present only when FEAT_AA32 is implemented and FEAT_ECV is implemented. Otherwise, direct accesses to CNTVCTSS are UNDEFINED.

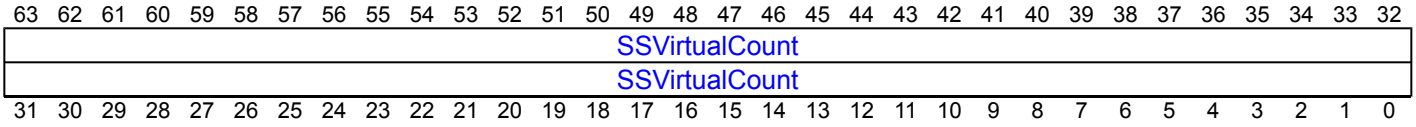
All reads to the CNTVCTSS occur in program order relative to reads to CNTVCT or CNTVCTSS.

This register is a view of the CNTVCT register for which reads appear to occur in program order relative to other instructions, without the need for any explicit synchronization. Reads of this register return a value consistent with the counter not being read until the read instruction is known to be non-speculative.

Attributes

CNTVCTSS is a 64-bit register.

Field descriptions



SSVirtualCount, bits [63:0]

Self-Synchronized Virtual count value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCTSS

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b1001

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_ECV)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && CNTKCTL.PL0VCTEN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif ELIsInHost(EL0) && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
(!EL2Enabled() || !ELIsInHost(EL0)) then
            R[t, t2] = PhysicalCountInt() - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) && !ELUsingAArch32(EL2) &&
(!EL2Enabled() || !ELIsInHost(EL0)) then
            R[t, t2] = PhysicalCountInt() - CNTVOFF;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            R[t, t2] = PhysicalCountInt() - CNTVOFF;
        else
            R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            R[t, t2] = PhysicalCountInt() - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            R[t, t2] = PhysicalCountInt() - CNTVOFF;
        else
            R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL2) then
        R[t, t2] = PhysicalCountInt() - CNTVOFF;
    else
        R[t, t2] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        R[t, t2] = PhysicalCountInt() - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        R[t, t2] = PhysicalCountInt() - CNTVOFF;
    else
        R[t, t2] = PhysicalCountInt();
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVOFF, Counter-timer Virtual Offset register

The CNTVOFF characteristics are:

Purpose

Holds the 64-bit virtual offset. This is the offset between the physical count value visible in CNTPCT and the virtual count value visible in CNTVCT.

Configuration

AArch32 System register CNTVOFF bits [63:0] are architecturally mapped to AArch64 System register CNTVOFF_EL2[63:0].

This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to CNTVOFF are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3 and the virtual counter uses a fixed virtual offset of zero.

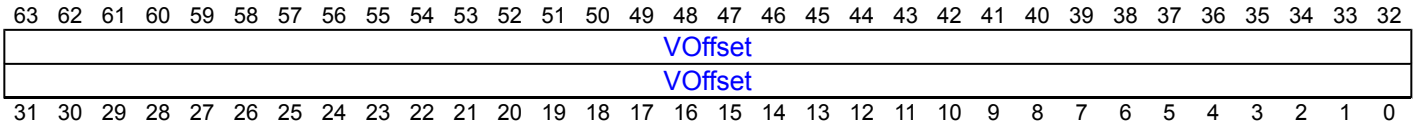
Note

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the virtual counter uses a fixed virtual offset of zero when CNTVCT is read from Non-secure EL0.

Attributes

CNTVOFF is a 64-bit register.

Field descriptions



VOffset, bits [63:0]

Virtual offset.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVOFF

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t, t2] = CNTVOFF;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t, t2] = CNTVOFF;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1110	0b0100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTVOFF = R[t, t2];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTVOFF = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CONTEXTIDR, Context ID Register

The CONTEXTIDR characteristics are:

Purpose

Identifies the current Process Identifier and, when using the Short-descriptor translation table format, the Address Space Identifier.

The value of the whole of this register is called the Context ID and is used by:

- The debug logic, for Linked and Unlinked Context ID matching.
- The trace logic, to identify the current process.

The significance of this register is for debug and trace use only.

Configuration

This register is banked between CONTEXTIDR and CONTEXTIDR_S and CONTEXTIDR_NS.

AArch32 System register CONTEXTIDR bits [31:0] are architecturally mapped to AArch64 System register CONTEXTIDR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to CONTEXTIDR are UNDEFINED.

The register format depends on whether address translation is using the Long-descriptor or the Short-descriptor translation table format.

Attributes

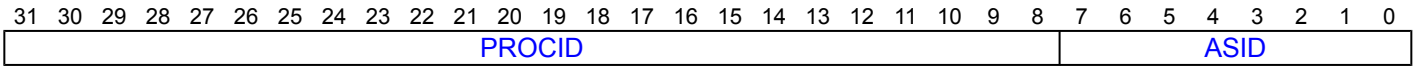
CONTEXTIDR is a 32-bit register.

This register has the following instances:

- CONTEXTIDR, when EL3 is not implemented or FEAT_AA64 is implemented.
- CONTEXTIDR_S, when FEAT_AA32EL3 is implemented.
- CONTEXTIDR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When TTBCR.EAE == 0:



PROCID, bits [31:8]

Process Identifier. This field must be programmed with a unique value that identifies the current process.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

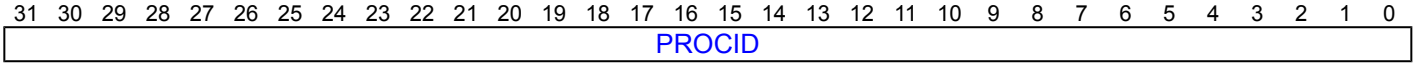
ASID, bits [7:0]

Address Space Identifier. This field is programmed with the value of the current ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When TTBCR.EAE == 1:



PROCID, bits [31:0]

Process Identifier. This field must be programmed with a unique value that identifies the current process.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CONTEXTIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = CONTEXTIDR_NS;
    else
        R[t] = CONTEXTIDR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = CONTEXTIDR_NS;
    else
        R[t] = CONTEXTIDR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = CONTEXTIDR_S;
    else
        R[t] = CONTEXTIDR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CONTEXTIDR_NS = R[t];
    else
        CONTEXTIDR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CONTEXTIDR_NS = R[t];
    else
        CONTEXTIDR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CONTEXTIDR_S = R[t];
    else
        CONTEXTIDR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

COSPRCTX, Clear Other Speculative Prediction Restriction by Context

The COSPRCTX characteristics are:

Purpose

Clear Other Speculative Prediction Restriction by Context applies to prediction resources not managed by other speculation restriction System instructions.

The actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control any predictions occurring after the instruction is complete and synchronized.

This instruction applies to all speculative access except:

- Cache Prefetch predictions.
- Control Flow predictions.
- Data Value predictions.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the PE that executed the original restriction instruction, and a subsequent Context Synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of Cache Allocation Resources so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations, the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

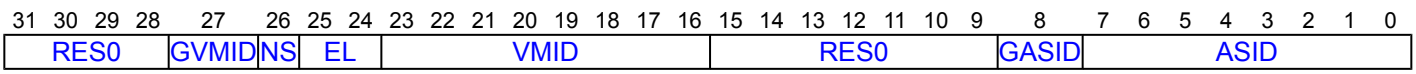
Configuration

This instruction is present only when FEAT_AA32 is implemented and FEAT_SPECRES2 is implemented. Otherwise, direct accesses to COSPRCTX are UNDEFINED.

Attributes

COSPRCTX is a 32-bit System instruction.

Field descriptions



Bits [31:28]

Reserved, RES0.

GVMID, bit [27]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 target execution context.
0b1	Applies to all VMIDs for an EL0 or EL1 target execution context.

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

NS, bit [26]

Security State.

NS	Meaning
0b0	Secure state.
0b1	Non-secure state.

If the instruction is executed in Non-secure state, this field is treated as 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

If the instruction is executed at an Exception level lower than the specified level, or is specified to apply to a combination of Exception level and Security state that is not implemented, this instruction is treated as a NOP.

VMID, bits [23:16]

Only applies when bit[27] is 0 and the target execution context is either:

- EL1.
- EL0 when EL2 is using AArch32 or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0, this field is treated as the current VMID if any of the following are true:

- EL2 is using AArch32.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

Bits [15:9]

Reserved, RES0.

GASID, bit [8]

Execution of this instruction applies to all ASIDs, or a specified ASID.

GASID	Meaning
0b0	Applies to specified ASID for an EL0 target execution context.
0b1	Applies to all ASIDs for an EL0 target execution context.

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [7:0]

Only applies to an EL0 target execution context and when bit[8] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

Executing COSPRCTX

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0011	0b110

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_SPECRES2)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && SCTLR.EnRCTX == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T7
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.COSPRCTX == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnRCTX == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch32.RestrictPrediction(R[t], RestrictType_Other);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x03);
    else
        AArch32.RestrictPrediction(R[t], RestrictType_Other);
elsif PSTATE.EL == EL2 then
    AArch32.RestrictPrediction(R[t], RestrictType_Other);
elsif PSTATE.EL == EL3 then
    AArch32.RestrictPrediction(R[t], RestrictType_Other);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CP15DMB, Data Memory Barrier System instruction

The CP15DMB characteristics are:

Purpose

Performs a Data Memory Barrier.

Arm deprecates any use of this System instruction, and strongly recommends that software use the DMB instruction instead.

Configuration

This instruction is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CP15DMB are UNDEFINED.

Attributes

CP15DMB is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing CP15DMB

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1010	0b101

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
SCTLR_EL1.CP15BEN == '0' then
        UNDEFINED;
    elsif ELIsInHost(EL0) && SCTLR_EL2.CP15BEN == '0' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T7
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        CP15DMB();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DMB();
elsif PSTATE.EL == EL2 then
    if HSCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DMB();
elsif PSTATE.EL == EL3 then
    if SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DMB();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CP15DSB, Data Synchronization Barrier System instruction

The CP15DSB characteristics are:

Purpose

Performs a Data Synchronization Barrier.

Arm deprecates any use of this System instruction, and strongly recommends that software use the DSB instruction instead.

Configuration

This instruction is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CP15DSB are UNDEFINED.

Attributes

CP15DSB is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing CP15DSB

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1010	0b100

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
SCTLR_EL1.CP15BEN == '0' then
        UNDEFINED;
    elsif ELIsInHost(EL0) && SCTLR_EL2.CP15BEN == '0' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T7
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        CP15DSB();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DSB();
elsif PSTATE.EL == EL2 then
    if HSCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DSB();
elsif PSTATE.EL == EL3 then
    if SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DSB();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CP15ISB, Instruction Synchronization Barrier System instruction

The CP15ISB characteristics are:

Purpose

Performs an Instruction Synchronization Barrier.

Arm deprecates any use of this System instruction, and strongly recommends that software use the ISB instruction instead.

Configuration

This instruction is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CP15ISB are UNDEFINED.

Attributes

CP15ISB is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing CP15ISB

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
SCTLR_EL1.CP15BEN == '0' then
        UNDEFINED;
    elsif ELIsInHost(EL0) && SCTLR_EL2.CP15BEN == '0' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T7
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        CP15ISB();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15ISB();
elsif PSTATE.EL == EL2 then
    if HSCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15ISB();
elsif PSTATE.EL == EL3 then
    if SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15ISB();
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPACR, Architectural Feature Access Control Register

The CPACR characteristics are:

Purpose

Controls access to trace, and to Advanced SIMD and floating-point functionality from EL0, EL1, and EL3.

In an implementation that includes EL2, the CPACR has no effect on instructions executed at EL2.

Configuration

AArch32 System register CPACR bits [31:0] are architecturally mapped to AArch64 System register CPACR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to CPACR are UNDEFINED.

Bits in the NSACR control Non-secure access to the CPACR fields. See the field descriptions for more information.

Note

In the register field descriptions, controls are described as applying at specified Privilege levels. This is because, in Secure state, a PL1 control:

- Applies to execution in a Secure EL3 mode when EL3 is using AArch32.
- Applies to execution in a Secure EL1 mode when EL3 is using AArch64.

See 'Security state, Exception levels, and AArch32 execution privilege'.

Attributes

CPACR is a 32-bit register.

Field descriptions

31	30 29	28	27 26 25 24	23 22	21 20	19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ASEDIS	RES0	TRCDIS	RES0	cp11	cp10	RES0

ASEDIS, bit [31]

Disables PL0 and PL1 execution of Advanced SIMD instructions.

ASEDIS	Meaning
0b0	This control permits execution of Advanced SIMD instructions at PL0 and PL1.
0b1	All instruction encodings that are Advanced SIMD instruction encodings, but are not also floating-point instruction encodings, are UNDEFINED at PL0 and PL1.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0. Otherwise, it is IMPLEMENTATION DEFINED whether this field is implemented as a RW field. If it is not implemented as a RW field, it is RAZ/WI.

If EL3 is implemented and is using AArch32, and the value of NSACR.NSASEDIS is 1, this field behaves as RAO/WI in Non-secure state, regardless of its actual value. This applies even if the field is implemented as RAZ/WI.

For the list of instructions affected by this field, see 'Controls of Advanced SIMD operation that do not apply to floating-point operation'.

See the description of CPACR.cp10 for a list of other controls that can disable or trap execution of Advanced SIMD instructions in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Bits [30:29]

Reserved, RES0.

TRCDIS, bit [28]

Traps PL0 and PL1 System register accesses to all implemented trace registers to Undefined mode.

TRCDIS	Meaning
`0b0`	This control has no effect on PL0 and PL1 System register accesses to trace registers.
`0b1`	PL0 and PL1 System register accesses to all implemented trace registers are trapped to Undefined mode.

If the implementation does not include a trace unit, or does not include a System register interface to the trace unit registers, this field is RES0. Otherwise, it is IMPLEMENTATION DEFINED whether this field is implemented as a RW field. If it is not implemented as a RW field, it is RAZ/WI.

If EL3 is implemented and is using AArch32, and the value of NSACR.NSTRCDIS is 1, this field behaves as RAO/WI in Non-secure state, regardless of its actual value. This applies even if the field is implemented as RAZ/WI.

Note

- The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED.
- The Arm architecture does not provide traps on trace register accesses through the optional memory-mapped external debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:24]

Reserved, RES0.

cp11, bits [23:22]

The value of this field is ignored. If this field is programmed with a different value to the cp10 field then this field is UNKNOWN on a direct read of the CPACR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

In Non-secure state, if EL3 is implemented and is using AArch32, when the value of NSACR.cp10 is 0, this field behaves as RAZ/WI, regardless of its actual value.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'00'`.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - EL3 is using AArch32.
  - !IsCurrentSecurityState(SS_Secure).
  - NSACR.cp10 == 0.

cp10, bits [21:20]

Defines the access rights for the Advanced SIMD and floating-point functionality. Possible values of the field are:

cp10	Meaning
0b00	PL0 and PL1 accesses to Advanced SIMD and floating-point registers or instructions are UNDEFINED.
0b01	PL0 accesses to Advanced SIMD and floating-point registers or instructions are UNDEFINED.
0b10	Reserved. The effect of programming this field to this value is CONSTRAINED UNPREDICTABLE choice between: • The value of this field is treated as 0b00, 0b01, or 0b11 for all purposes other than reading the value of the field. • PL1 accesses to floating-point and Advanced SIMD regsiters or instructions are UNDEFINED.
0b11	This control permits full access to the Advanced SIMD and floating-point functionality from PL0 and PL1.

The Advanced SIMD and floating-point features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the FPSCR, FPSID, MVFR0, MVFR1, MVFR2, or FPEXC System registers.

Note

The CPACR has no effect on Advanced SIMD and floating-point accesses from PL2. These can be disabled by the HCPTR.TCP10 field.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

In Non-secure state, if EL3 is implemented and is using AArch32, when the value of NSACR.cp10 is 0, this field behaves as RAZ/WI, regardless of its actual value.

Execution of Advanced SIMD and floating-point instructions in AArch32 state can be disabled or trapped by the following controls:

- CPACR.cp10, or, if executing at EL0, CPACR_EL1.FPEN.
- FPEXC.EN.
- If executing in Non-secure state:
  - HCPTR.TCP10, or if EL2 is using AArch64, CPTR_EL2.TFP.
  - NSACR.cp10, or if EL3 is using AArch64, CPTR_EL3.TFP.
- For Advanced SIMD instructions only:
  - CPACR.ASEDIS.
  - If executing in Non-secure state, HCPTR.TASE and NSACR.NSASEDIS.

See the descriptions of the controls for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - EL3 is implemented.
  - EL3 is using AArch32.
  - !IsCurrentSecurityState(SS_Secure).
  - NSACR.cp10 == 0.

Bits [19:0]

Reserved, RES0.

Accessing CPACR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TCPAC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TCPAC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = CPACR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = CPACR;
elsif PSTATE.EL == EL3 then
    R[t] = CPACR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.TCPAC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCPTR.TCPAC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        CPACR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        CPACR = R[t];
elsif PSTATE.EL == EL3 then
    CPACR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPPRCTX, Cache Prefetch Prediction Restriction by Context

The CPPRCTX characteristics are:

Purpose

Cache Prefetch Prediction Restriction by Context applies to all Cache Allocation Resources that predict cache allocations based on information gathered within the target execution context or contexts.

The actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control cache prefetch predictions occurring after the instruction is complete and synchronized.

This instruction applies to all:

- Instruction caches.
- Data caches.
- TLB prefetching hardware used by the executing PE that applies to the supplied context or contexts.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of Cache Allocation Resources so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

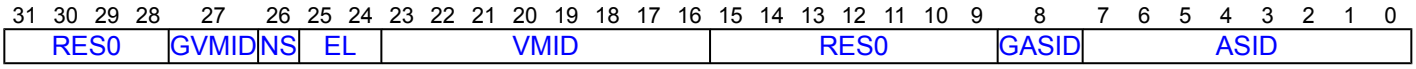
Configuration

This instruction is present only when FEAT_AA32 is implemented and FEAT_SPECRES is implemented. Otherwise, direct accesses to CPPRCTX are UNDEFINED.

Attributes

CPPRCTX is a 32-bit System instruction.

Field descriptions



Bits [31:28]

Reserved, RES0.

GVMID, bit [27]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 target execution context.
0b1	Applies to all VMIDs for an EL0 or EL1 target execution context.

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

NS, bit [26]

Security State.

NS	Meaning
0b0	Secure state.
0b1	Non-secure state.

If the instruction is executed in Non-secure state, this field is treated as 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

If the instruction is executed at an Exception level lower than the specified level, or is specified to apply to a combination of Exception level and Security state that is not implemented, this instruction is treated as a NOP.

VMID, bits [23:16]

Only applies when bit[27] is 0 and the target execution context is either:

- EL1.
- EL0 when EL2 is using AArch32 or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0, this field is treated as the current VMID if any of the following are true:

- EL2 is using AArch32.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

Bits [15:9]

Reserved, RES0.

GASID, bit [8]

Execution of this instruction applies to all ASIDs or a specified ASID.

GASID	Meaning
0b0	Applies to specified ASID for an EL0 target execution context.
0b1	Applies to all ASIDs for an EL0 target execution context.

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [7:0]

Only applies for an EL0 target execution context and when bit[8] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

Executing CPPRCTX

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0011	0b111

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_SPECRES)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && SCTLR.EnRCTX == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T7
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.CPPRCTX == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnRCTX == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch32.RestrictPrediction(R[t], RestrictType_CachePrefetch);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x03);
    else
        AArch32.RestrictPrediction(R[t], RestrictType_CachePrefetch);
elsif PSTATE.EL == EL2 then
    AArch32.RestrictPrediction(R[t], RestrictType_CachePrefetch);
elsif PSTATE.EL == EL3 then
    AArch32.RestrictPrediction(R[t], RestrictType_CachePrefetch);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPSR, Current Program Status Register

The CPSR characteristics are:

Purpose

Holds PE status and control information.

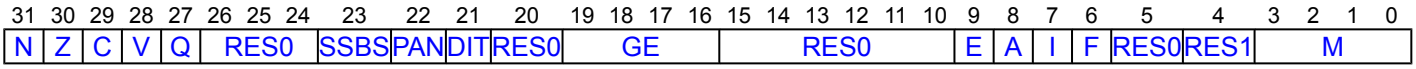
Configuration

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to CPSR are UNDEFINED.

Attributes

CPSR is a 32-bit register.

Field descriptions



N, bit [31]

Negative condition flag. Set to bit[31] of the result of the last flag-setting instruction. If the result is regarded as a two's complement signed integer, then N is set to 1 if the result was negative, and N is set to 0 if the result was positive or zero.

Z, bit [30]

Zero condition flag. Set to 1 if the result of the last flag-setting instruction was zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

C, bit [29]

Carry condition flag. Set to 1 if the last flag-setting instruction resulted in a carry condition, for example an unsigned overflow on an addition.

V, bit [28]

Overflow condition flag. Set to 1 if the last flag-setting instruction resulted in an overflow condition, for example a signed overflow on an addition.

Q, bit [27]

Cumulative saturation bit. Set to 1 to indicate that overflow or saturation occurred in some instructions.

Bits [26:24]

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass Safe.

Prohibits speculative loads or stores that might practically allow a cache timing side channel.

A speculative value in a register is used in a potentially speculatively exploitable manner if it is used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence or if the execution timing of any other instructions in the speculative sequence is a function of the data loaded under speculation.

SSBS	Meaning
0b0	Hardware is not permitted to use speculative register values in a potentially speculatively exploitable manner if the speculative read that loads the register is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.
0b1	When the value of PSTATE.SSBS is 1, hardware is permitted to use speculative register values in a potentially speculatively exploitable manner if the speculative read that loads the register is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.

The value of this bit is usually set to the value described by the SCTLR.DSSBS bit on exceptions to any mode except Hyp mode, and the value described by HSCTLR.DSSBS on exceptions to Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never.

PAN	Meaning
0b0	The translation system is the same as Armv8.0.
0b1	Disables privileged read and write accesses to addresses accessible at EL0.

The value of this bit is usually preserved on taking an exception, except in the following situations:

- When the target of the exception is EL1, and the value of the SCTLR.SPAN bit for the current Security state is 0, this bit is set to 1.
- When the target of the exception is EL3, from Secure state, and the value of the Secure SCTLR.SPAN is 0, this bit is set to 1.
- When the target of the exception is EL3, from Non-secure state, this bit is set to 0 regardless of the value of the Secure SCTLR.SPAN bit.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing.

DIT	Meaning
0b0	The architecture makes no statement about the timing properties of any instructions.
0b1	The architecture requires that: • The timing of every load and store instruction is insensitive to the value of the data being loaded or stored. • For certain data processing instructions, the instruction takes a time that is independent of: ◦ The values of the data supplied in any of its registers. ◦ The values of the NZCV flags. • For certain data processing instructions, the response of the instruction to asynchronous exceptions does not vary based on: ◦ The values of the data supplied in any of its registers. ◦ The values of the NZCV flags.

The Operational Information section of a data processing instruction description indicates if that instruction is affected by this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

GE, bits [19:16]

Greater than or Equal flags, for parallel addition and subtraction.

Bits [15:10]

Reserved, RES0.

E, bit [9]

Endianness state bit. Controls the load and store endianness for data accesses:

E	Meaning
0b0	Little-endian operation
0b1	Big-endian operation.

Instruction fetches ignore this bit.

If an implementation does not provide Big-endian support, this bit is RES0. If it does not provide Little-endian support, this bit is RES1.

If an implementation provides Big-endian support but only at EL0, this bit is RES0 for an exception return to any Exception level other than EL0.

Likewise, if it provides Little-endian support only at EL0, this bit is RES1 for an exception return to any Exception level other than EL0.

When the reset value of the SCTLR.EE bit is defined by a configuration input signal, that value also applies to the CPSR.E bit on reset, and therefore applies to software execution from reset.

A, bit [8]

SError exception mask bit.

A	Meaning
0b0	Exception not masked.
0b1	Exception masked.

I, bit [7]

IRQ mask bit.

I	Meaning
0b0	Exception not masked.
0b1	Exception masked.

F, bit [6]

FIQ mask bit.

F	Meaning
0b0	Exception not masked.
0b1	Exception masked.

Bit [5]

Reserved, RES0.

Bit [4]

Reserved, RES1.

M, bits [3:0]

Current PE mode.

M	Meaning
0b0000	User.
0b0001	FIQ.
0b0010	IRQ.
0b0011	Supervisor.
0b0110	Monitor.
0b0111	Abort.
0b1010	Hyp.
0b1011	Undefined.
0b1111	System.

Accessing CPSR

CPSR can be read using the MRS instruction and written using the MSR (register) or MSR (immediate) instructions.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CSSELR, Cache Size Selection Register

The CSSELR characteristics are:

Purpose

Selects the current Cache Size ID Register, CCSIDR, by specifying the required cache level and the cache type, which is either instruction cache or data cache.

If FEAT_CCIDX is implemented, CSSELR also selects the current CCSIDR2.

Configuration

This register is banked between CSSELR and CSSELR_S and CSSELR_NS.

AArch32 System register CSSELR bits [31:0] are architecturally mapped to AArch64 System register CSSELR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to CSSELR are UNDEFINED.

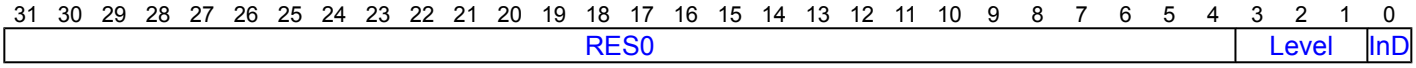
Attributes

CSSELR is a 32-bit register.

This register has the following instances:

- CSSELR, when EL3 is not implemented or FEAT_AA64 is implemented.
- CSSELR_S, when FEAT_AA32EL3 is implemented.
- CSSELR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



Bits [31:4]

Reserved, RES0.

Level, bits [3:1]

Cache level of required cache. Permitted values are:

Level	Meaning
0b000	Level 1 cache.
0b001	Level 2 cache.
0b010	Level 3 cache.
0b011	Level 4 cache.
0b100	Level 5 cache.
0b101	Level 6 cache.
0b110	Level 7 cache.

All other values are reserved.

If CSSELR.{Level, InD} is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

InD, bit [0]

Instruction not Data bit. Permitted values are:

InD	Meaning
0b0	Data or unified cache.
0b1	Instruction cache.

If CSSELR.{Level, InD} is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CSSELR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b010	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_EVT) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_EVT) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = CSSELR_NS;
    else
        R[t] = CSSELR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = CSSELR_NS;
    else
        R[t] = CSSELR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = CSSELR_S;
    else
        R[t] = CSSELR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b010	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_EVT) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_EVT) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CSSELR_NS = R[t];
    else
        CSSELR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        CSSELR_NS = R[t];
    else
        CSSELR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CSSELR_S = R[t];
    else
        CSSELR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTR, Cache Type Register

The CTR characteristics are:

Purpose

Provides information about the architecture of the caches.

Configuration

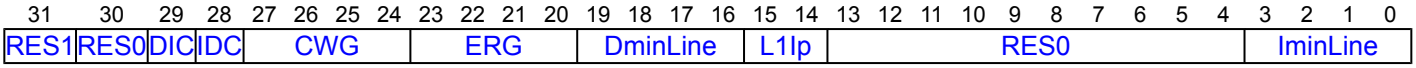
AArch32 System register CTR bits [31:0] are architecturally mapped to AArch64 System register CTR_EL0[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to CTR are UNDEFINED.

Attributes

CTR is a 32-bit register.

Field descriptions



Bit [31]

Reserved, RES1.

Bit [30]

Reserved, RES0.

DIC, bit [29]

Instruction cache invalidation requirements for data to instruction coherence.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DIC	Meaning
0b0	Instruction cache invalidation to the Point of Unification is required for data to instruction coherence.
0b1	Instruction cache invalidation to the Point of Unification is not required for data to instruction coherence.

All PEs in the same Inner Shareable shareability domain must have a common value of this field.

Access to this field is **RO**.

IDC, bit [28]

Data cache clean requirements for instruction to data coherence. The meaning of this bit is:

The value of this field is an IMPLEMENTATION DEFINED choice of:

IDC	Meaning
0b0	Data cache clean to the Point of Unification is required for instruction to data coherence, unless CLIDR.LoC == 0b000 or (CLIDR.LoUIS == 0b000 and CLIDR.LoUU == 0b000).
0b1	Data cache clean to the Point of Unification is not required for instruction to data coherence.

If CTR.DIC is 1, then the value reported in this field must be 1.

The Effective value of IDC is 1 if any of the following are true:

- CTR.IDC == 1.
- CLIDR.LoC == 0b000.
- CLIDR.LoUIS == 0b000 and CLIDR.LoUU == 0b000.

All PEs in the same Inner Shareable shareability domain must have a common Effective value of IDC.

Access to this field is **RO**.

CWG, bits [27:24]

Cache writeback granule. Log_2 of the number of words of the maximum size of memory that can be overwritten as a result of the eviction of a cache entry that has had a memory location in it modified.

A value of 0b0000 indicates that this register does not provide Cache writeback granule information and either:

- The architectural maximum of 512 words (2KB) must be assumed.
- The Cache writeback granule can be determined from maximum cache line size encoded in the Cache Size ID Registers.

Values greater than 0b1001 are reserved.

Arm recommends that an implementation that does not support cache write-back implements this field as 0b0001. This applies, for example, to an implementation that supports only write-through caches.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ERG, bits [23:20]

Exclusives reservation granule. Log_2 of the number of words of the maximum size of the reservation granule that has been implemented for the Load-Exclusive and Store-Exclusive instructions.

The use of the value 0b0000 is deprecated.

The value 0b0001 and values greater than 0b1001 are reserved.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

DminLine, bits [19:16]

Log_2 of the number of words in the smallest cache line of all the data caches and unified caches that are controlled by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

L1Ip, bits [15:14]

Level 1 instruction cache policy. Indicates the indexing and tagging policy for the L1 instruction cache.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1Ip	Meaning
0b00	Reserved.
0b01	ASID-tagged Virtual Index, Virtual Tag (AIVIVT).
0b10	Virtual Index, Physical Tag (VIPT).
0b11	Physical Index, Physical Tag (PIPT).

From Armv8.0, the value 0b01 is not permitted.

Access to this field is **RO**.

Bits [13:4]

Reserved, RES0.

IminLine, bits [3:0]

Log_2 of the number of words in the smallest cache line of all the instruction caches that are controlled by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = CTR;
elsif PSTATE.EL == EL2 then
    R[t] = CTR;
elsif PSTATE.EL == EL3 then
    R[t] = CTR;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DACR, Domain Access Control Register

The DACR characteristics are:

Purpose

Defines the access permission for each of the sixteen memory domains.

Configuration

This register is banked between DACR and DACR_S and DACR_NS.

AArch32 System register DACR bits [31:0] are architecturally mapped to AArch64 System register DACR32_EL2[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DACR are UNDEFINED.

This register has no function when TTBCR.EAE is set to 1, to select the Long-descriptor translation table format.

Attributes

DACR is a 32-bit register.

This register has the following instances:

- DACR, when EL3 is not implemented or FEAT_AA64 is implemented.
- DACR_S, when FEAT_AA32EL3 is implemented.
- DACR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

D<n>, bits [2n+1:2n], for n = 15 to 0

Domain n access permission, where n = 0 to 15. Permitted values are:

D<n>	Meaning
0b00	No access. Any access to the domain generates a Domain fault.
0b01	Client. Accesses are checked against the permission bits in the translation tables.
0b11	Manager. Accesses are not checked against the permission bits in the translation tables.

The value 0b10 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DACR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0011	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T3
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = DACR_NS;
    else
        R[t] = DACR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = DACR_NS;
    else
        R[t] = DACR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = DACR_S;
    else
        R[t] = DACR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0011	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T3
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        DACR_NS = R[t];
    else
        DACR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        DACR_NS = R[t];
    else
        DACR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            DACR_S = R[t];
        else
            DACR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGAUTHSTATUS, Debug Authentication Status register

The DBGAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

Configuration

AArch32 System register DBGAUTHSTATUS bits [31:0] are architecturally mapped to AArch64 System register DBGAUTHSTATUS_EL1[31:0].

AArch32 System register DBGAUTHSTATUS bits [31:0] are architecturally mapped to External register DBGAUTHSTATUS_EL1[31:0].

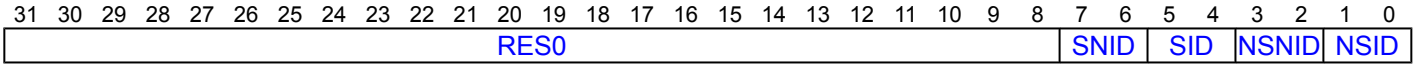
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGAUTHSTATUS are UNDEFINED.

This register is required in all implementations.

Attributes

DBGAUTHSTATUS is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SNID, bits [7:6]

When FEAT_Debugv8p4 is implemented:

Secure Non-Invasive Debug.

This field has the same value as DBGAUTHSTATUS.SID.

Otherwise:

Secure Non-Invasive Debug.

SNID	Meaning
0b00	Secure state is not implemented.
0b10	Implemented and disabled. ExternalSecureNoninvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalSecureNoninvasiveDebugEnabled() == TRUE.

All other values are reserved.

SID, bits [5:4]

Secure Invasive Debug.

SID	Meaning
0b00	Secure state is not implemented.
0b10	Implemented and disabled. ExternalSecureInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalSecureInvasiveDebugEnabled() == TRUE.

All other values are reserved.

NSNID, bits [3:2]

When FEAT_Debugv8p4 is implemented:

Non-secure Non-invasive debug.

NSNID	Meaning
0b00	Non-secure state is not implemented.
0b11	Implemented and enabled. EL3 is implemented or the Effective value of SCR.NS is 1.

All other values are reserved.

Otherwise:

Non-secure Non-Invasive Debug.

NSNID	Meaning
0b00	Non-secure state is not implemented.
0b10	Implemented and disabled. ExternalNoninvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE.

All other values are reserved.

NSID, bits [1:0]

Non-secure Invasive Debug.

NSID	Meaning
0b00	Non-secure state is not implemented.
0b10	Implemented and disabled. ExternalInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalInvasiveDebugEnabled() == TRUE.

All other values are reserved.

Accessing DBGAUTHSTATUS

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0111	0b1110	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGAUTHSTATUS;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGAUTHSTATUS;
elsif PSTATE.EL == EL3 then
    R[t] = DBGAUTHSTATUS;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGBCR<n>, Debug Breakpoint Control Registers, n = 0 - 15

The DBGBCR<n> characteristics are:

Purpose

Holds control information for a breakpoint. Forms breakpoint n together with value register DBGBVR<n>. If EL2 is implemented and this breakpoint supports Context matching, DBGBVR<n> can be associated with a Breakpoint Extended Value Register DBGBXVR<n> for VMID matching.

Configuration

AArch32 System register DBGBCR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGBCR<n>_EL1[31:0].

AArch32 System register DBGBCR<n> bits [31:0] are architecturally mapped to External register DBGBCR<n>_EL1[31:0].

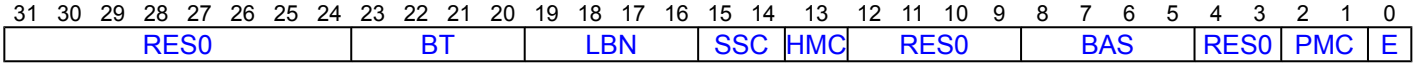
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGBCR<n> are UNDEFINED.

If breakpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGBCR<n> is a 32-bit register.

Field descriptions



When the E field is zero, all the other fields in the register are ignored.

Bits [31:24]

Reserved, RES0.

BT, bits [23:20]

Breakpoint Type. Possible values are:

BT	Meaning
0b0000	Unlinked instruction address match. DBGBVR<n> is the address of an instruction.
0b0001	As 0b0000, but linked to a Context matching breakpoint.
0b0010	Unlinked Context ID match. If the Effective value of HCR_EL2.E2H is 1, and either the PE is executing at EL0 with HCR_EL2.TGE set to 1 or the PE is executing at EL2, then DBGBVR<n>.ContextID is compared against CONTEXTIDR_EL2. Otherwise, DBGBVR<n>.ContextID is compared against CONTEXTIDR.
0b0011	As 0b0010 with linking enabled.
0b0100	Unlinked instruction address mismatch. DBGBVR<n> is the address of an instruction to be stepped.
0b0101	As 0b0100, but linked to a Context matching breakpoint.
0b0110	Unlinked CONTEXTIDR match. DBGBVR<n>.ContextID is a Context ID compared against CONTEXTIDR.
0b0111	As 0b0110 with linking enabled.
0b1000	Unlinked VMID match. DBGBXVR<n>.VMID is a VMID compared against VTTBR.VMID.
0b1001	As 0b1000 with linking enabled.
0b1010	Unlinked VMID and Context ID match. DBGBVR<n>.ContextID is a Context ID compared against CONTEXTIDR, and DBGBXVR<n>.VMID is a VMID compared against VTTBR.VMID.
0b1011	As 0b1010 with linking enabled.
0b1100	Unlinked CONTEXTIDR_EL2 match. DBGBXVR<n>.ContextID2 is a Context ID compared against CONTEXTIDR_EL2.
0b1101	As 0b1100 with linking enabled.
0b1110	Unlinked Full Context ID match. DBGBVR<n>.ContextID is compared against CONTEXTIDR, and DBGBXVR<n>.ContextID2 is compared against CONTEXTIDR_EL2.
0b1111	As 0b1110 with linking enabled.

For more information on Breakpoints and their constraints, see 'Breakpoint exceptions' and 'Reserved DBGBCR<n>.BT values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked Breakpoint Number. For Linked address matching breakpoints, this specifies the index of the Context-matching breakpoint linked to.

For all other breakpoint types, this field is ignored and reads of the register return an UNKNOWN value.

This field is ignored when the value of DBGBCR<n>.E is 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the HMC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields.

For more information, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' and 'Reserved DBGBCR<n>.{SSC, HMC, PMC} values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the SSC, bits [15:14] description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [12:9]

Reserved, RES0.

BAS, bits [8:5]

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state.

The permitted values depend on the breakpoint type.

For Address match breakpoints, the permitted values are:

BAS	Match instruction at	Constraint for debuggers
0b0011	DBGBVR<n>	Use for T32 instructions
0b1100	DBGBVR<n>+2	Use for T32 instructions
0b1111	DBGBVR<n>	Use for A32 instructions

All other values are reserved. For more information, see 'Reserved DBGBCR<n>.BAS values'.

For more information on using the BAS field in Address Match breakpoints, see 'Using the BAS field in Address Match breakpoints'.

For Address mismatch breakpoints in an AArch32 stage 1 translation regime, the permitted values are:

BAS	Step instruction at	Constraint for debuggers
0b0000	-	Use for a match anywhere breakpoint
0b0011	DBGBVR<n>	Use for T32 instructions
0b1100	DBGBVR<n>+2	Use for T32 instructions
0b1111	DBGBVR<n>	Use for A32 instructions

All other values are reserved. For more information, see 'Reserved DBGBCR<n>.BAS values'.

For more information on using the BAS field in address mismatch breakpoints, see 'Using the BAS field in Address Match breakpoints'.

For Context matching breakpoints, this field is RES1 and ignored.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

PMC, bits [2:1]

Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and HMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the DBGBCR<n>.SSC description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable breakpoint DBGBVR<n>. Possible values are:

E	Meaning
0b0	Breakpoint disabled.
0b1	Breakpoint enabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBCR<n>

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	m[3:0]	0b101

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif m >= NUM_BREAKPOINTS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGBCR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGBCR[m];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGBCR[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	m[3:0]	0b101

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif m >= NUM_BREAKPOINTS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBCR[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBCR[m] = R[t];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBCR[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGBVR<n>, Debug Breakpoint Value Registers, n = 0 - 15

The DBGBVR<n> characteristics are:

Purpose

Holds a value for use in breakpoint matching, either the virtual address of an instruction or a context ID. Forms breakpoint n together with control register DBGBCR<n>. If EL2 is implemented and this breakpoint supports Context matching, DBGBVR<n> can be associated with a Breakpoint Extended Value Register DBGBXVR<n> for VMID matching.

Configuration

AArch32 System register DBGBVR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGBVR<n>_EL1[31:0].

AArch32 System register DBGBVR<n> bits [31:0] are architecturally mapped to External register DBGBVR<n>_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGBVR<n> are UNDEFINED.

How this register is interpreted depends on the value of DBGBCR<n>.BT.

- When DBGBCR<n>.BT is `0b0x0x`, this register holds a virtual address.
- When DBGBCR<n>.BT is `0bxx1x`, this register holds a Context ID.

For other values of DBGBCR<n>.BT, this register is RES0.

Some breakpoints might not support Context ID comparison. For more information, see the description of the DBGDIDR.CTX_CMPs field.

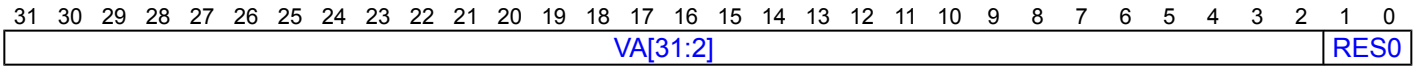
If breakpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGBVR<n> is a 32-bit register.

Field descriptions

When DBGBCR<n>.BT IN {0b0x0x}:



VA[31:2], bits [31:2]

Bits[31:2] of the address value for comparison.

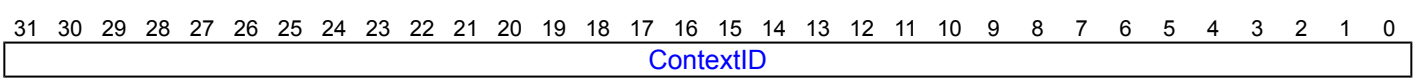
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When DBGBCR<n>.BT IN {0b001x}:



ContextID, bits [31:0]

Context ID value for comparison.

The value is compared against CONTEXTIDR_EL2 when all of the following are true:

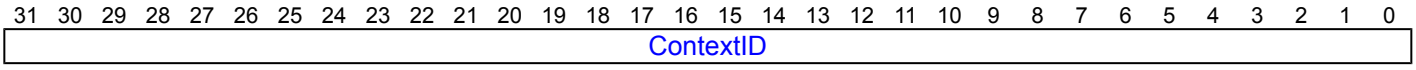
- The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.
- The PE is executing at EL0.

Otherwise, the value is compared against CONTEXTIDR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>.BT IN {0b101x} and EL2 is implemented:



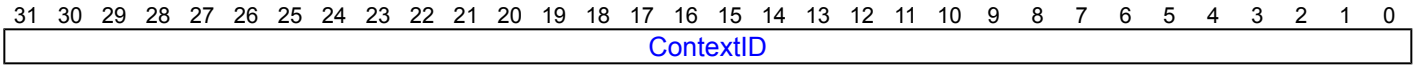
ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>.BT IN {0bx11x}, EL2 is implemented, and FEAT_Debugv8p1 is implemented:



ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBVR<n>

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	m[3:0]	0b100

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif m >= NUM_BREAKPOINTS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGBVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGBVR[m];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGBVR[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	m[3:0]	0b100

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif m >= NUM_BREAKPOINTS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBVR[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBVR[m] = R[t];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBVR[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGBXVR<n>, Debug Breakpoint Extended Value Registers, n = 0 - 15

The DBGBXVR<n> characteristics are:

Purpose

Holds a value for use in breakpoint matching, to support VMID matching. Used in conjunction with a control register DBGBCR<n> and a value register DBGBVR<n>, where EL2 is implemented and breakpoint n supports Context matching.

Configuration

AArch32 System register DBGBXVR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGBVR<n>_EL1[63:32].

AArch32 System register DBGBXVR<n> bits [31:0] are architecturally mapped to External register DBGBVR<n>_EL1[63:32].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGBXVR<n> are UNDEFINED.

How this register is interpreted depends on the value of DBGBCR<n>.BT.

- When DBGBCR<n>.BT is `0b10xx`, this register holds a VMID.
- When DBGBCR<n>.BT is `0b11xx`, this register holds a Context ID.

For other values of DBGBCR<n>.BT, this register is RES0.

Accesses to this register are UNDEFINED in any of the following cases:

- Breakpoint n is not implemented.
- Breakpoint n does not support Context matching.
- EL2 is not implemented.

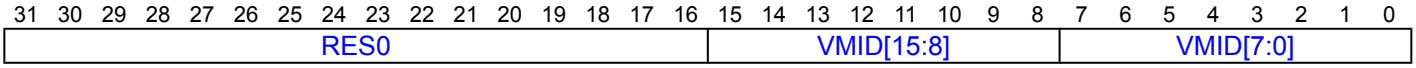
For more information, see the description of the DBGDIDR.CTX_CMPs field.

Attributes

DBGBXVR<n> is a 32-bit register.

Field descriptions

When DBGBCR<n>.BT IN {0b10xx} and EL2 is implemented:



Bits [31:16]

Reserved, RES0.

VMID[15:8], bits [15:8]

When FEAT_VMID16 is implemented and VTCR_EL2.VS == 1:

Extension to VMID[7:0]. For more information, see VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [7:0]

VMID value for comparison. The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>.BT IN {0b11xx} and EL2 is implemented:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ContextID2																															

ContextID2, bits [31:0]

When FEAT_Debugv8p1 is implemented:

Context ID value for comparison against CONTEXTIDR_EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing DBGBXVR<n>

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0001	m[3:0]	0b001

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif m >= NUM_BREAKPOINTS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGBXVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGBXVR[m];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGBXVR[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0001	m[3:0]	0b001

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif m >= NUM_BREAKPOINTS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBXVR[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBXVR[m] = R[t];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBXVR[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMCLR, Debug CLAIM Tag Clear register

The DBGCLAIMCLR characteristics are:

Purpose

Used by software to read the values of the CLAIM tag bits, and to clear CLAIM tag bits to 0.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMSET register.

Configuration

AArch32 System register DBGCLAIMCLR bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMCLR_EL1[31:0].

AArch32 System register DBGCLAIMCLR bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMSET_EL1[31:0].

AArch32 System register DBGCLAIMCLR bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMSET[31:0].

AArch32 System register DBGCLAIMCLR bits [31:0] are architecturally mapped to External register DBGCLAIMCLR_EL1[31:0].

AArch32 System register DBGCLAIMCLR bits [31:0] are architecturally mapped to External register DBGCLAIMSET_EL1[31:0].

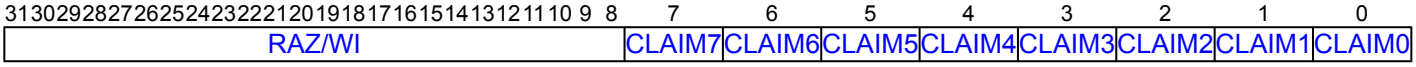
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGCLAIMCLR are UNDEFINED.

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMCLR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RAZ/WI.

CLAIM<m>, bit [m], for m = 7 to 0

Claim Tag Clear. Indicates the current status of Claim Tag bit <m>, and is used to clear Claim Tag bit <m> to 0.

CLAIM<m>	Meaning
0b0	On a read: Claim Tag bit <m> is not set. On a write: Ignored.
0b1	On a read: Claim Tag bit <m> is set. On a write: Clear Claim tag bit <m> to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Access to this field is **W1C**.

Accessing DBGCLAIMCLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0111	0b1001	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGCLAIMCLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGCLAIMCLR;
elsif PSTATE.EL == EL3 then
    R[t] = DBGCLAIMCLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0111	0b1001	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGCLAIMCLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGCLAIMCLR = R[t];
elsif PSTATE.EL == EL3 then
    DBGCLAIMCLR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMSET, Debug CLAIM Tag Set register

The DBGCLAIMSET characteristics are:

Purpose

Used by software to set the CLAIM tag bits to 1.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMCLR register.

Configuration

AArch32 System register DBGCLAIMSET bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMSET_EL1[31:0].

AArch32 System register DBGCLAIMSET bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMCLR_EL1[31:0].

AArch32 System register DBGCLAIMSET bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMCLR[31:0].

AArch32 System register DBGCLAIMSET bits [31:0] are architecturally mapped to External register DBGCLAIMSET_EL1[31:0].

AArch32 System register DBGCLAIMSET bits [31:0] are architecturally mapped to External register DBGCLAIMCLR_EL1[31:0].

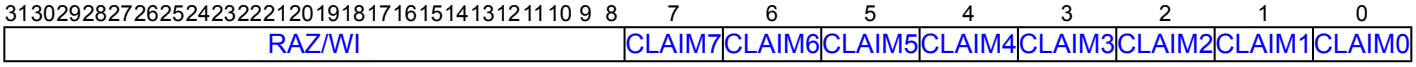
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGCLAIMSET are UNDEFINED.

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMSET is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RAZ/WI.

CLAIM<m>, bit [m], for m = 7 to 0

Claim Tag Set. Used to set Claim Tag bit <m> to 1.

CLAIM<m>	Meaning
0b0	On a write: Ignored.
0b1	On a write: Set Claim Tag bit <m> to 1.

Access to this field is **RAO/W1S**.

Accessing DBGCLAIMSET

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0111	0b1000	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGCLAIMSET;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGCLAIMSET;
elsif PSTATE.EL == EL3 then
    R[t] = DBGCLAIMSET;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0111	0b1000	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGCLAIMSET = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGCLAIMSET = R[t];
elsif PSTATE.EL == EL3 then
    DBGCLAIMSET = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDCCINT, DCC Interrupt Enable Register

The DBGDCCINT characteristics are:

Purpose

Enables interrupt requests to be signaled based on the DCC status flags.

Configuration

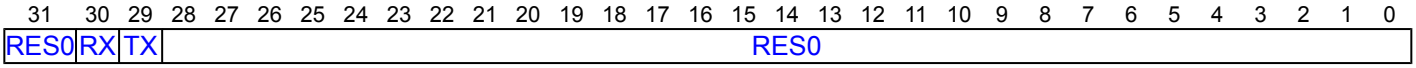
AArch32 System register DBGDCCINT bits [31:0] are architecturally mapped to AArch64 System register MDCCINT_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGDCCINT are UNDEFINED.

Attributes

DBGDCCINT is a 32-bit register.

Field descriptions



Bit [31]

Reserved, RES0.

RX, bit [30]

DCC interrupt request enable control for DTRRX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

RX	Meaning
0b0	No interrupt request generated by DTRRX.
0b1	Interrupt request will be generated on RXfull == 1.

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TX, bit [29]

DCC interrupt request enable control for DTRTX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

TX	Meaning
0b0	No interrupt request generated by DTRTX.
0b1	Interrupt request will be generated on TXfull == 0.

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [28:0]

Reserved, RES0.

Accessing DBGDCCINT

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    R[t] = DBGDCCINT;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDCCINT;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDCCINT;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = DBGDCCINT;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    DBGDCCINT = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDCCINT = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDCCINT = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        DBGDCCINT = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDEVID, Debug Device ID register 0

The DBGDEVID characteristics are:

Purpose

Adds to the information given by the DBGDIDR by describing other features of the debug implementation.

Configuration

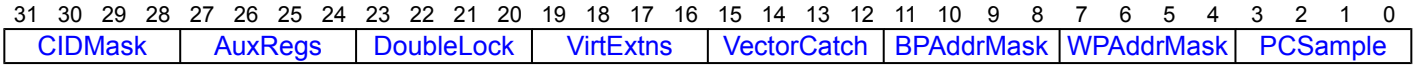
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGDEVID are UNDEFINED.

This register is required in all implementations.

Attributes

DBGDEVID is a 32-bit register.

Field descriptions



CIDMask, bits [31:28]

Indicates the level of support for the Context ID matching breakpoint masking capability.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CIDMask	Meaning
0b0000	Context ID masking is not implemented.
0b0001	Context ID masking is implemented.

All other values are reserved. The value of this for Armv8 is 0b0000.

Access to this field is **RO**.

AuxRegs, bits [27:24]

Indicates support for Auxiliary registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AuxRegs	Meaning
0b0000	None supported.
0b0001	Support for External Debug Auxiliary Control Register, EDACR.

All other values are reserved.

Access to this field is **RO**.

DoubleLock, bits [23:20]

OS Double Lock implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DoubleLock	Meaning
0b0000	OS Double Lock is not implemented. DBGOSDLR is RAZ/WI.
0b0001	OS Double Lock is implemented. DBGOSDLR is RW.

FEAT_DoubleLock implements the functionality identified by the value 0b0001.

All other values are reserved.

Access to this field is **RO**.

VirtExtns, bits [19:16]

Indicates whether EL2 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VirtExtns	Meaning
0b0000	EL2 is not implemented.
0b0001	EL2 is implemented.

All other values are reserved.

Access to this field is **RO**.

VectorCatch, bits [15:12]

Defines the form of Vector Catch exception implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VectorCatch	Meaning
0b0000	Address matching Vector Catch exception implemented.
0b0001	Exception matching Vector Catch exception implemented.

All other values are reserved.

Access to this field is **RO**.

BPAddrMask, bits [11:8]

Indicates the level of support for the instruction address matching breakpoint masking capability.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BPAddrMask	Meaning
0b0000	Breakpoint address masking might be implemented. If not implemented, DBGBCR<n>[28:24] is RAZ/WI.
0b0001	Breakpoint address masking is implemented.
0b1111	Breakpoint address masking is not implemented. DBGBCR<n>[28:24] is RES0.

All other values are reserved. The value of this for Armv8 is 0b1111.

Access to this field is **RO**.

WPAddrMask, bits [7:4]

Indicates the level of support for the data address matching watchpoint masking capability.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WPAddrMask	Meaning
0b0000	Watchpoint address masking might be implemented. If not implemented, DBGWCR<n>.MASK (Address mask) is RAZ/WI.
0b0001	Watchpoint address masking is implemented.
0b1111	Watchpoint address masking is not implemented. DBGWCR<n>.MASK (Address mask) is RES0.

All other values are reserved. The value of this for Armv8 is 0b0001.

Access to this field is **RO**.

PCSample, bits [3:0]

Indicates the level of PC Sample-based Profiling support using external debug registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PCSample	Meaning
0b0000	PC Sample-based Profiling Extension is not implemented in the external debug registers space.
0b0010	Only EDPCSR and EDCIDSR are implemented. This option is only permitted if EL3 and EL2 are not implemented.
0b0011	EDPCSR, EDCIDSR, and EDVIDSR are implemented.

All other values are reserved.

When FEAT_PCSRv8p2 is implemented, the only permitted value is 0b0000.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of PMDEVID.PCSample.

Access to this field is **RO**.

Accessing DBGDEVID

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0111	0b0010	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDEVID;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDEVID;
elsif PSTATE.EL == EL3 then
    R[t] = DBGDEVID;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDEVID1, Debug Device ID register 1

The DBGDEVID1 characteristics are:

Purpose

Adds to the information given by the DBGDIDR by describing other features of the debug implementation.

Configuration

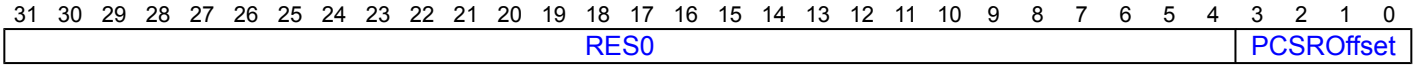
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGDEVID1 are UNDEFINED.

This register is required in all implementations.

Attributes

DBGDEVID1 is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

PCSROffset, bits [3:0]

This field indicates the offset applied to PC samples returned by reads of EDPCSR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PCSROffset	Meaning
0b0000	EDPCSR is not implemented.
0b0010	EDPCSR implemented. Samples have no offset applied and do not sample the instruction set state in AArch32 state.

When FEAT_PCSRv8p2 is implemented, the only permitted value is 0b0000.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of PMDEVID.PCSample.

Access to this field is **RO**.

Accessing DBGDEVID1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1110	0b000	0b0111	0b0001	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDEVID1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDEVID1;
elsif PSTATE.EL == EL3 then
    R[t] = DBGDEVID1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDEVID2, Debug Device ID register 2

The DBGDEVID2 characteristics are:

Purpose

Reserved for future descriptions of features of the debug implementation.

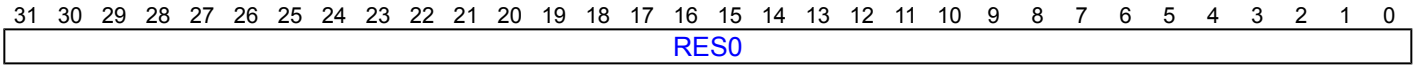
Configuration

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGDEVID2 are UNDEFINED.

Attributes

DBGDEVID2 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing DBGDEVID2

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0111	0b0000	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDEVID2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDEVID2;
elsif PSTATE.EL == EL3 then
    R[t] = DBGDEVID2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDIDR, Debug ID Register

The DBGDIDR characteristics are:

Purpose

Specifies which version of the Debug architecture is implemented, and some features of the debug implementation.

Configuration

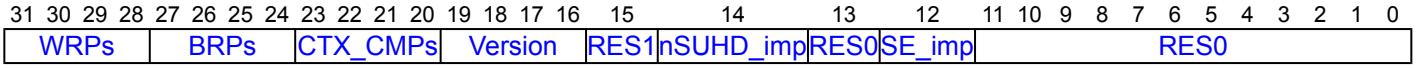
This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to DBGDIDR are UNDEFINED.

If EL1 cannot use AArch32 then the implementation of this register is OPTIONAL and deprecated.

Attributes

DBGDIDR is a 32-bit register.

Field descriptions



WRPs, bits [31:28]

Number of watchpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WRPs	Meaning
0b0001..0b1111	The number of watchpoints, minus 1.

If FEAT_Debugv8p9 is implemented and 16 or more watchpoints are implemented, this field reads as 0b1111.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 watchpoints.

The value 0b0000 is reserved.

Access to this field is **RO**.

BRPs, bits [27:24]

Number of breakpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BRPs	Meaning
0b0001..0b1111	The number of breakpoints, minus 1.

If FEAT_Debugv8p9 is implemented and 16 or more breakpoints are implemented, this field reads as 0b1111.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 breakpoints.

The value 0b0000 is reserved.

Access to this field is **RO**.

CTX_CMPs, bits [23:20]

Number of context-aware breakpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CTX_CMPs	Meaning
0b0000..0b1111	The number of context-aware breakpoints, minus 1.

The value of this field is never greater than DBGDIDR.BRPs.

If FEAT_Debugv8p9 is implemented and 16 or more context-aware breakpoints are implemented, this field reads as 0b1111.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 breakpoints.

Access to this field is **RO**.

Version, bits [19:16]

Debug architecture version. Indicates presence of Armv8 debug architecture.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Version	Meaning
0b0000	Not supported.
0b0001	Armv6, v6 Debug architecture, with System registers access.
0b0010	Armv6, v6.1 Debug architecture, with System registers access.
0b0011	Armv7, v7 Debug architecture, with only baseline System registers.
0b0100	Armv7, v7 Debug architecture, with all System registers implemented.
0b0101	Armv7, v7.1 Debug architecture, with System registers access.
0b0110	Armv8.0 debug architecture.
0b0111	Armv8.1 debug architecture, FEAT_Debugv8p1.
0b1000	Armv8.2 debug architecture, FEAT_Debugv8p2.
0b1001	Armv8.4 debug architecture, FEAT_Debugv8p4.
0b1010	Armv8.8 debug architecture, FEAT_Debugv8p8.
0b1011	Armv8.9 debug architecture, FEAT_Debugv8p9.

All other values are reserved.

From Armv8.0, the values 0b0000, 0b0001, 0b0010, 0b0011, 0b0100, and 0b0101 are not permitted.

FEAT_Debugv8p1 implements the functionality identified by the value 0b0111.

FEAT_Debugv8p2 implements the functionality identified by the value 0b1000.

FEAT_Debugv8p4 implements the functionality identified by the value 0b1001.

FEAT_Debugv8p8 implements the functionality identified by the value 0b1010.

FEAT_Debugv8p9 implements the functionality identified by the value 0b1011.

From Armv8.1, when FEAT_Debugv8p1 is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

From Armv8.9, the value 0b1010 is not permitted.

Access to this field is **RO**.

Bit [15]

Reserved, RES1.

nSUHD_imp, bit [14]

Previously indicated that Secure User Halting Debug is not implemented.

The value of this field must match the value of the SE_imp field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [13]

Reserved, RES0.

SE_imp, bit [12]

EL3 implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SE_imp	Meaning
0b0	EL3 not implemented.
0b1	EL3 implemented.

The value of this field must match the value of the nSUHD_imp field.

Access to this field is **RO**.

Bits [11:0]

Reserved, RES0.

Accessing DBGDIDR

Arm deprecates any access to this register from EL0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    R[t] = DBGDIDR;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
(HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && (HCR.TGE
== '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDIDR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDIDR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDIDR;
elsif PSTATE.EL == EL3 then
```

```
    R[t] = DBGDIDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDRAR, Debug ROM Address Register

The DBGDRAR characteristics are:

Purpose

Defines the base physical address of a 4KB-aligned memory-mapped debug component, usually a ROM table that locates and describes the memory-mapped debug components in the system. Use of this register is deprecated.

Configuration

AArch32 System register DBGDRAR bits [63:0] are architecturally mapped to AArch64 System register MDRAR_EL1[63:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to DBGDRAR are UNDEFINED.

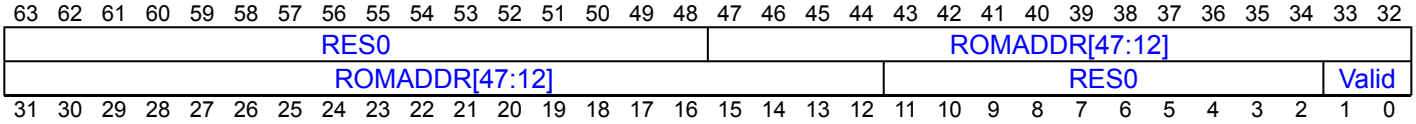
DBGDRAR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, bits [31:0] are read.

If EL1 cannot use AArch32 then the implementation of this register is OPTIONAL and deprecated.

Attributes

DBGDRAR is a 64-bit register.

Field descriptions



Bits [63:48]

Reserved, RES0.

ROMADDR[47:12], bits [47:12]

Bits[47:12] of the ROM table physical address.

If the physical address size in bits (PAsize) is less than 48 then the register bits corresponding to ROMADDR [47:PAsize] are RES0.

Bits [11:0] of the ROM table physical address are zero.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system where the implementation only supports execution in AArch32 state.

If DBGDRAR.Valid == `0b00`, then this field is UNKNOWN.

In an implementation that includes EL3, ROMADDR is an address in Non-secure memory. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure memory.

Bits [11:2]

Reserved, RES0.

Valid, bits [1:0]

This field indicates whether the ROM Table address is valid.

Valid	Meaning
0b00	ROM Table address is not valid. Software must ignore ROMADDR.
0b11	ROM Table address is valid.

Other values are reserved.

Arm recommends implementations set this field to zero.

Accessing DBGDRAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    R[t] = DBGDRAR<31:0>;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
(HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && (HCR.TGE
== '1' || HDCR.<TDE,TDRA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDRAR<31:0>;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDRA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDRA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDRAR<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDRAR<31:0>;
elsif PSTATE.EL == EL3 then
```

```
    R[t] = DBGDRAR<31:0>;
```

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1110	0b0001	0b0000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    R[t, t2] = DBGDRAR;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x0C);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
(HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && (HCR.TGE
== '1' || HDCR.<TDE,TDRA> != '00') then
        AArch32.TakeHypTrapException(0x0C);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
    else
        R[t, t2] = DBGDRAR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDRA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDRA> != '00' then
        AArch32.TakeHypTrapException(0x0C);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
    else
        R[t, t2] = DBGDRAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
    else
        R[t, t2] = DBGDRAR;
elsif PSTATE.EL == EL3 then
```

```
    R[t, t2] = DBGDRAR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDSAR, Debug Self Address Register

The DBGDSAR characteristics are:

Purpose

In earlier versions of the Arm Architecture, this register defines the offset from the base address defined in DBGDRAR of the physical base address of the debug registers for the PE. Use of this register is deprecated.

Configuration

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to DBGDSAR are UNDEFINED.

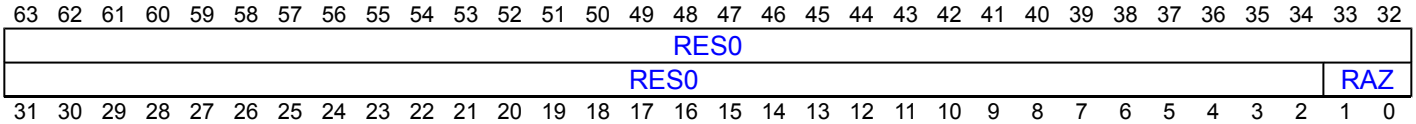
DBGDSAR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, bits [31:0] are read.

If EL1 cannot use AArch32 then the implementation of this register is OPTIONAL and deprecated.

Attributes

DBGDSAR is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

Bits [1:0]

Reserved, RAZ.

This field indicates whether the debug self address offset is valid. For ARMv8, this field is always `0b00`, the offset is not valid.

Accessing DBGDSAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    R[t] = DBGDSAR<31:0>;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
(HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && (HCR.TGE
== '1' || HDCR.<TDE,TDRA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDSAR<31:0>;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDRA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDRA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDSAR<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDSAR<31:0>;
elsif PSTATE.EL == EL3 then
```

```
    R[t] = DBGDSAR<31:0>;
```

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1110	0b0010	0b0000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    R[t, t2] = DBGDSAR;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x0C);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
(HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && (HCR.TGE
== '1' || HDCR.<TDE,TDRA> != '00') then
        AArch32.TakeHypTrapException(0x0C);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
    else
        R[t, t2] = DBGDSAR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDRA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDRA> != '00' then
        AArch32.TakeHypTrapException(0x0C);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
    else
        R[t, t2] = DBGDSAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
    else
        R[t, t2] = DBGDSAR;
elsif PSTATE.EL == EL3 then
```

```
    R[t, t2] = DBGDSAR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDSCRext, Debug Status and Control Register, External View

The DBGDSCRext characteristics are:

Purpose

Main control register for the debug implementation.

Configuration

AArch32 System register DBGDSCRext bits [31:0] are architecturally mapped to AArch64 System register MDSCR_EL1[31:0].

AArch32 System register DBGDSCRext bits [31:29, 27:26, 23:21, 19, 14, 6] are architecturally mapped to External register EDSCR[31:29, 27:26, 23:21, 19, 14, 6].

AArch32 System register DBGDSCRext bit [15] is architecturally mapped to AArch32 System register DBGDSCRint[15].

AArch32 System register DBGDSCRext bit [12] is architecturally mapped to AArch32 System register DBGDSCRint[12].

AArch32 System register DBGDSCRext bits [5:2] are architecturally mapped to AArch32 System register DBGDSCRint[5:2].

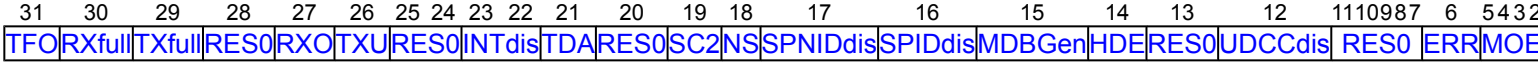
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGDSCRext are UNDEFINED.

This register is required in all implementations.

Attributes

DBGDSCRext is a 32-bit register.

Field descriptions



TFO, bit [31]

When FEAT_TRF is implemented:

Trace Filter override. Used for save/restore of EDSCR.TFO.

When the OS Lock is unlocked, DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When the OS Lock is locked, DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.TFO. Reads and writes of this bit are indirect accesses to EDSCR.TFO.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

Otherwise:

Reserved, RES0.

RXfull, bit [30]

DTRRX full. Used for save/restore of EDSCR.RXfull.

When DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.RXfull. Reads and writes of this bit are indirect accesses to EDSCR.RXfull.

Arm deprecates use of this bit other than for save/restore. Use DBGDSCRint to access the DTRRX full status.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

TXfull, bit [29]

DTRTX full. Used for save/restore of EDSCR.TXfull.

When DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.TXfull. Reads and writes of this bit are indirect accesses to EDSCR.TXfull.

Arm deprecates use of this bit other than for save/restore. Use DBGDSCRint to access the DTRTX full status.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

Bit [28]

Reserved, RES0.

RXO, bit [27]

Used for save/restore of EDSCR.RXO.

When DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.RXO. Reads and writes of this bit are indirect accesses to EDSCR.RXO.

When DBGOSLSR.OSLK == 1, if bits [27,6] of the value written to DBGDSCRext are {1,0}, that is, the RXO bit is 1 and the ERR bit is 0, the PE sets EDSCR.{RXO,ERR} to UNKNOWN values.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

TXU, bit [26]

Used for save/restore of EDSCR.TXU.

When DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.TXU. Reads and writes of this bit are indirect accesses to EDSCR.TXU.

When DBGOSLSR.OSLK == 1, if bits [26,6] of the value written to DBGDSCRext are {1,0}, that is, the TXU bit is 1 and the ERR bit is 0, the PE sets EDSCR.{TXU,ERR} to UNKNOWN values.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

Bits [25:24]

Reserved, RES0.

INTdis, bits [23:22]

Used for save/restore of EDSCR.INTdis.

When DBGOSLSR.OSLK == 0, software must treat it as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this field holds the value of EDSCR.INTdis. Reads and writes of this field are indirect accesses to EDSCR.INTdis.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'00'`.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

TDA, bit [21]

Used for save/restore of EDSCR.TDA.

When DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.TDA. Reads and writes of this bit are indirect accesses to EDSCR.TDA.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

Bit [20]

Reserved, RES0.

SC2, bit [19]

When FEAT_PCSRv8 is implemented, FEAT_VHE is implemented, and FEAT_PCSRv8p2 is not implemented:

Used for save/restore of EDSCR.SC2.

When DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.SC2. Reads and writes of this bit are indirect accesses to EDSCR.SC2.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

Otherwise:

Reserved, RES0.

NS, bit [18]

Non-secure status.

Arm deprecates use of this field.

NS	Meaning
`0b0`	Secure state.
`0b1`	Non-secure state.

Access to this field is **RO**.

SPNIDdis, bit [17]

When EL3 is implemented:

Secure privileged profiling disabled status bit.

SPNIDdis	Meaning
`0b0`	Profiling allowed in Secure privileged modes.
`0b1`	Profiling prohibited in Secure privileged modes.

This field reads as 0 if any of the following applies, and reads as 1 otherwise:

- FEAT_Debugv8p2 is not implemented and `ExternalSecureNoninvasiveDebugEnabled()` returns TRUE.
- EL3 is using AArch32 and the value of SDCR.SPME is 1.
- EL3 is using AArch64 and the value of MDCR_EL3.SPME is 1.

Arm deprecates use of this field.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

SPIDdis, bit [16]

When EL3 is implemented:

Secure privileged AArch32 invasive self-hosted debug disabled status bit. The value of this bit depends on the value of SDCR.SPD and the pseudocode function `AArch32.SelfHostedSecurePrivilegedInvasiveDebugEnabled()`.

SPIDdis	Meaning
`0b0`	Self-hosted debug enabled in Secure privileged AArch32 modes.
`0b1`	Self-hosted debug disabled in Secure privileged AArch32 modes.

This bit reads as 1 if any of the following is true and reads as 0 otherwise:

- EL3 is using AArch32 and SDCR.SPD has the value `0b10`.
- EL3 is using AArch64 and MDCR_EL3.SPD32 has the value `0b10`.

- EL3 is using AArch32, SDCR.SPD has the value `0b00`, and `AArch32.SelfHostedSecurePrivilegedInvasiveDebugEnabled()` returns FALSE.
- EL3 is using AArch64, MDCR_EL3.SPD32 has the value `0b00`, and `AArch32.SelfHostedSecurePrivilegedInvasiveDebugEnabled()` returns FALSE.

Arm deprecates use of this field.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

MDBGen, bit [15]

Monitor debug events enable. Enable Breakpoint, Watchpoint, and Vector Catch exceptions.

MDBGen	Meaning
`0b0`	Breakpoint, Watchpoint, and Vector Catch exceptions disabled.
`0b1`	Breakpoint, Watchpoint, and Vector Catch exceptions enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

HDE, bit [14]

Used for save/restore of EDSCR.HDE.

When DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.HDE. Reads and writes of this bit are indirect accesses to EDSCR.HDE.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

Bit [13]

Reserved, RES0.

UDCCdis, bit [12]

Traps EL0 accesses to the DCC registers to Undefined mode.

UDCCdis	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 accesses to the DBGDSCRint, DBGDTRRXint, DBGDTRTXint, DBGDIDR, DBGDSAR, and DBGDRAR are trapped to Undefined mode.

Note

All accesses to these registers are trapped, including LDC and STC accesses to DBGDTRTXint and DBGDTRRXint, and MRRC accesses to DBGDSAR and DBGDRAR.

Traps of EL0 accesses to the DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [11:7]

Reserved, RES0.

ERR, bit [6]

Used for save/restore of EDSCR.ERR.

When DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.ERR. Reads and writes of this bit are indirect accesses to EDSCR.ERR.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

MOE, bits [5:2]

Method of Entry for debug exception. When a debug exception is taken to an Exception level using AArch32, this field is set to indicate the event that caused the exception:

MOE	Meaning
0b0001	Breakpoint.
0b0011	Software breakpoint (BKPT) instruction.
0b0101	Vector catch.
0b1010	Watchpoint.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

Accessing DBGDSCRext

Individual fields within this register might have restricted accessibility when the OS Lock is unlocked, DBGOSLSR.OSLK == 0. See the field descriptions for more detail.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDSCRext;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDSCRext;
elsif PSTATE.EL == EL3 then
    R[t] = DBGDSCRext;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDSCRext = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDSCRext = R[t];
elsif PSTATE.EL == EL3 then
    DBGDSCRext = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDSCRint, Debug Status and Control Register, Internal View

The DBGDSCRint characteristics are:

Purpose

Main control register for the debug implementation. This is an internal, read-only view.

Configuration

AArch32 System register DBGDSCRint bits [30:29] are architecturally mapped to AArch64 System register MDCCSR_EL0[30:29].

AArch32 System register DBGDSCRint bits [30:29] are architecturally mapped to External register EDSCR[30:29].

AArch32 System register DBGDSCRint bit [15] is architecturally mapped to AArch64 System register MDSCR_EL1[15].

AArch32 System register DBGDSCRint bit [12] is architecturally mapped to AArch64 System register MDSCR_EL1[12].

AArch32 System register DBGDSCRint bits [5:2] are architecturally mapped to AArch64 System register MDSCR_EL1[5:2].

AArch32 System register DBGDSCRint bit [15] is architecturally mapped to AArch32 System register DBGDSCRext[15].

AArch32 System register DBGDSCRint bit [12] is architecturally mapped to AArch32 System register DBGDSCRext[12].

AArch32 System register DBGDSCRint bits [5:2] are architecturally mapped to AArch32 System register DBGDSCRext[5:2].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to DBGDSCRint are UNDEFINED.

This register is required in all implementations.

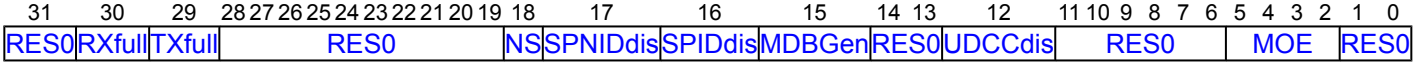
DBGDSCRint.{NS, SPNIDdis, SPIDdis, MDBGen, UDCCdis, MOE} are UNKNOWN when the register is accessed at EL0. However, although these values are not accessible at EL0 by instructions that are neither UNPREDICTABLE nor return UNKNOWN values, it is permissible for an implementation to return the values of DBGDSCRext.{NS, SPNIDdis, SPIDdis, MDBGen, UDCCdis, MOE} for these fields at EL0.

It is also permissible for an implementation to return the same values as defined for a read of DBGDSCRint at EL1 or above. (This is the case even if the implementation does not support AArch32 at EL1 or above.)

Attributes

DBGDSCRint is a 32-bit register.

Field descriptions



Bit [31]

Reserved, RES0.

RXfull, bit [30]

DTRRX full. Read-only view of the equivalent bit in the EDSCR.

TXfull, bit [29]

DTRTX full. Read-only view of the equivalent bit in the EDSCR.

Bits [28:19]

Reserved, RES0.

NS, bit [18]

Non-secure status.

Read-only view of the equivalent bit in the DBGDSCRext. Arm deprecates use of this field.

SPNIDdis, bit [17]

Secure privileged non-invasive debug disable.

Read-only view of the equivalent bit in the DBGDSCRext. Arm deprecates use of this field.

SPIDdis, bit [16]

Secure privileged invasive debug disable.

Read-only view of the equivalent bit in the DBGDSCRext. Arm deprecates use of this field.

MDBGen, bit [15]

Monitor debug events enable.

Read-only view of the equivalent bit in the DBGDSCRext.

Bits [14:13]

Reserved, RES0.

UDCCdis, bit [12]

User mode access to Debug Communications Channel disable.

Read-only view of the equivalent bit in the DBGDSCRext. Arm deprecates use of this field.

Bits [11:6]

Reserved, RES0.

MOE, bits [5:2]

Method of Entry for debug exception. When a debug exception is taken to an Exception level using AArch32, this field is set to indicate the event that caused the exception:

MOE	Meaning
0b0001	Breakpoint
0b0011	Software breakpoint (BKPT) instruction
0b0101	Vector catch
0b1010	Watchpoint

Read-only view of the equivalent bit in the DBGDSCRext.

Bits [1:0]

Reserved, RES0.

Accessing DBGDSCRint

When <Rt> is APSR_nzcv, encoded as R15, then instead of reading the entire register, the access copies DBGDSCRint[31:28] into the PSTATE NZCV flags.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    if t == 15 then
        ConstrainUnpredictableProcedure(Unpredictable_MRC_APSR_TARGET);
    else
        R[t] = DBGDSCRint;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
(HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && (HCR.TGE
== '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        if t == 15 then
            if Halted() then
                ConstrainUnpredictableProcedure(Unpredictable_MRC_APSR_TARGET);
            else
                PSTATE.<N,Z,C,V> = DBGDSCRint<31:28>;
        else
```

```
            R[t] = DBGDSCRint;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        if t == 15 then
            if Halted() then
                ConstrainUnpredictableProcedure(Unpredictable_MRC_APSR_TARGET);
            else
                PSTATE.<N,Z,C,V> = DBGDSCRint<31:28>;
        else
            R[t] = DBGDSCRint;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```

```
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        if t == 15 then
            if Halted() then
                ConstrainUnpredictableProcedure(Unpredictable_MRC_APSR_TARGET);
            else
                PSTATE.<N,Z,C,V> = DBGDSCRint<31:28>;
        else
            R[t] = DBGDSCRint;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        if t == 15 then
            if Halted() then
                ConstrainUnpredictableProcedure(Unpredictable_MRC_APSR_TARGET);
            else
                PSTATE.<N,Z,C,V> = DBGDSCRint<31:28>;
        else
            R[t] = DBGDSCRint;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRRXext, Debug OS Lock Data Transfer Register, Receive, External View

The DBGDTRRXext characteristics are:

Purpose

Used for save/restore of DBGDTRRXint. It is a component of the Debug Communications Channel.

Configuration

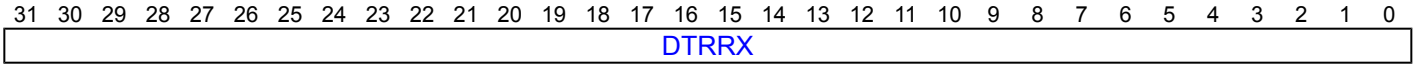
AArch32 System register DBGDTRRXext bits [31:0] are architecturally mapped to AArch64 System register OSDTRRX_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGDTRRXext are UNDEFINED.

Attributes

DBGDTRRXext is a 32-bit register.

Field descriptions



DTRRX, bits [31:0]

Update DTRRX without side-effect.

Writes to this register update the value in DTRRX and do not change RXfull.

Reads of this register return the last value written to DTRRX and do not change RXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRRXext

Arm deprecates reads and writes of DBGDTRRXext through the System register interface when the OS Lock is unlocked, DBGOSLSR.OSLK == 0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    R[t] = DBGDTRRXext;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDTRRXext;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDTRRXext;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = DBGDTRRXext;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    DBGDTRRXext = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRRXext = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRRXext = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        DBGDTRRXext = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRRXint, Debug Data Transfer Register, Receive

The DBGDTRRXint characteristics are:

Purpose

Transfers data from an external debugger to the PE. For example, it is used by a debugger transferring commands and data to a debug target. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communications Channel.

Configuration

AArch32 System register DBGDTRRXint bits [31:0] are architecturally mapped to AArch64 System register DBGDTRRX_EL0[31:0].

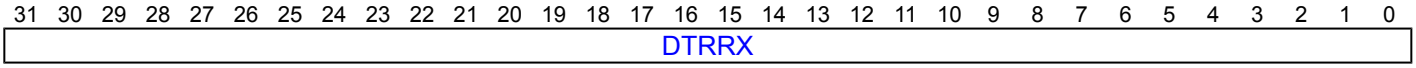
AArch32 System register DBGDTRRXint bits [31:0] are architecturally mapped to External register DBGDTRRX_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to DBGDTRRXint are UNDEFINED.

Attributes

DBGDTRRXint is a 32-bit register.

Field descriptions



DTRRX, bits [31:0]

Update DTRRX.

Reads of this register:

- If RXfull is 1, return the last value written to DTRRX.
- If RXfull is 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRRXint

Data can be stored to memory from this register using STC.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif Halted() then
    R[t] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
(HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && (HCR.TGE
== '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
```

```
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = Read_DBGDTR_EL0(32);
```

```
STC{<c>}{<q>} <coproc>, <CRd>, <addressing_mode>
```

coproc	CRd
0b1110	0b0101

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif Halted() then
    MemA[address, 4] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x06);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x06);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x06);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x06);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x06);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
(HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x06);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && (HCR.TGE
== '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x06);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    else
        MemA[address, 4] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x06);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x06);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x06);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x06);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    else
        MemA[address, 4] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
```

```
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    else
        MemA[address, 4] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        MemA[address, 4] = Read_DBGDTR_EL0(32);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRTXext, Debug OS Lock Data Transfer Register, Transmit

The DBGDTRTXext characteristics are:

Purpose

Used for save/restore of DBGDTRTXint. It is a component of the Debug Communication Channel.

Configuration

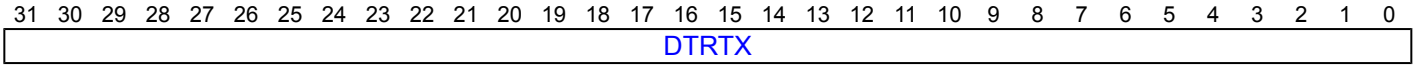
AArch32 System register DBGDTRTXext bits [31:0] are architecturally mapped to AArch64 System register OSDTRTX_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGDTRTXext are UNDEFINED.

Attributes

DBGDTRTXext is a 32-bit register.

Field descriptions



DTRTX, bits [31:0]

Return DTRTX without side-effect.

Reads of this register return the value in DTRTX and do not change TXfull.

Writes of this register update the value in DTRTX and do not change TXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRTXext

Arm deprecates reads and writes of DBGDTRTXext through the System register interface when the OS Lock is unlocked, DBGOSLSR.OSLK == 0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    R[t] = DBGDTRTXext;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDTRTXext;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGDTRTXext;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = DBGDTRTXext;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    DBGDTRTXext = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRTXext = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRTXext = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        DBGDTRTXext = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRTXint, Debug Data Transfer Register, Transmit

The DBGDTRTXint characteristics are:

Purpose

Transfers data from the PE to an external debugger. For example, it is used by a debug target to transfer data to the debugger. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communication Channel.

Configuration

AArch32 System register DBGDTRTXint bits [31:0] are architecturally mapped to AArch64 System register DBGDTRTX_EL0[31:0].

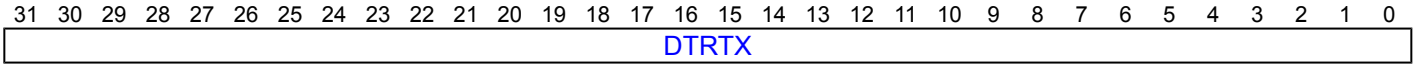
AArch32 System register DBGDTRTXint bits [31:0] are architecturally mapped to External register DBGDTRTX_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to DBGDTRTXint are UNDEFINED.

Attributes

DBGDTRTXint is a 32-bit register.

Field descriptions



DTRTX, bits [31:0]

DTRTX. Writes to this register:

- If TXfull is 1, DTRTX is set to an UNKNOWN value.
- If TXfull is 0, update the value in DTRTX.

After the write, TXfull is set to 1.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRTXint

Data can be loaded from memory into this register using 'LDC (immediate)' and 'LDC (literal)'.

Accesses to this register use the following encodings in the System register encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif Halted() then
    Write_DBGDTR_EL0(R[t]);
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
(HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && (HCR.TGE
== '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        Write_DBGDTR_EL0(R[t]);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        Write_DBGDTR_EL0(R[t]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
```

```
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        Write_DBGDTR_EL0(R[t]);
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        Write_DBGDTR_EL0(R[t]);
```

```
LDC{<c>}{<q>} <coproc>, <CRd>, <addressing_mode>
```

coproc	CRd
0b1110	0b0101

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif Halted() then
    Write_DBGDTR_EL0(MemA[address, 4]);
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x06);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x06);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x06);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x06);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x06);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
(HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x06);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && (HCR.TGE
== '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x06);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    else
        Write_DBGDTR_EL0(MemA[address, 4]);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x06);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TDCC
== '1' then
        AArch32.TakeHypTrapException(0x06);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x06);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x06);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    else
        Write_DBGDTR_EL0(MemA[address, 4]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
```

```
IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TDCC ==
'1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x06);
    else
        Write_DBGDTR_EL0(MemA[address, 4]);
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        Write_DBGDTR_EL0(MemA[address, 4]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGOSDLR, Debug OS Double Lock Register

The DBGOSDLR characteristics are:

Purpose

Locks out the external debug interface.

Configuration

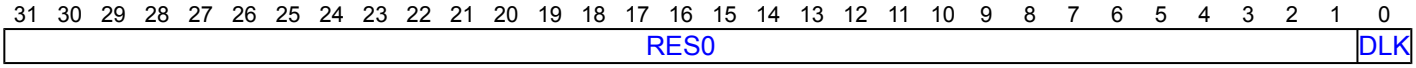
AArch32 System register DBGOSDLR bits [31:0] are architecturally mapped to AArch64 System register OSDLR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGOSDLR are UNDEFINED.

Attributes

DBGOSDLR is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

DLK, bit [0]

When FEAT_DoubleLock is implemented:

OS Double Lock control bit.

DLK	Meaning
0b0	OS Double Lock unlocked.
0b1	OS Double Lock locked, if DBGPRCR.CORENPDRQ (Core no powerdown request) bit is set to 0 and the PE is in Non-debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RAZ/WI.

Accessing DBGOSDLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0001	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDOSA> != '00' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
IMPLEMENTATION_DEFINED "Trapped by MDCR_EL2.TDOSA") then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDOSA> != '00' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
IMPLEMENTATION_DEFINED "Trapped by HDCR.TDOSA") then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED
"Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGOSDLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED
"Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGOSDLR;
elsif PSTATE.EL == EL3 then
    R[t] = DBGOSDLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0001	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDOSA> != '00' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
IMPLEMENTATION_DEFINED "Trapped by MDCR_EL2.TDOSA") then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDOSA> != '00' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
IMPLEMENTATION_DEFINED "Trapped by HDCR.TDOSA") then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED
"Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGOSDLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED
"Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGOSDLR = R[t];
elsif PSTATE.EL == EL3 then
    DBGOSDLR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGOSECCR, Debug OS Lock Exception Catch Control Register

The DBGOSECCR characteristics are:

Purpose

Provides a mechanism for an operating system to access the contents of EDECCR that are otherwise invisible to software, so it can save/restore the contents of EDECCR over powerdown on behalf of the external debugger.

Configuration

AArch32 System register DBGOSECCR bits [31:0] are architecturally mapped to AArch64 System register OSECCR_EL1[31:0].

AArch32 System register DBGOSECCR bits [31:0] are architecturally mapped to External register EDECCR[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGOSECCR are UNDEFINED.

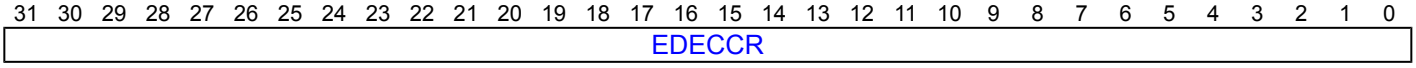
If DBGOSLSR.OSLK == 0 then DBGOSECCR returns an UNKNOWN value on reads and ignores writes.

Attributes

DBGOSECCR is a 32-bit register.

Field descriptions

When DBGOSLSR.OSLK == 1:



EDECCR, bits [31:0]

Used for save/restore to EDECCR over powerdown.

Reads or writes to this field are indirect accesses to EDECCR.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'00000000000000000000000000000000'`.

Accessing DBGOSECCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0110	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' then
        R[t] = bits(32) UNKNOWN;
    else
        R[t] = DBGOSECCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' then
        R[t] = bits(32) UNKNOWN;
    else
        R[t] = DBGOSECCR;
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' then
        R[t] = bits(32) UNKNOWN;
    else
        R[t] = DBGOSECCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0110	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' then
        return;
    else
        DBGOSECCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' then
        return;
    else
        DBGOSECCR = R[t];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' then
        return;
    else
        DBGOSECCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGOSLAR, Debug OS Lock Access Register

The DBGOSLAR characteristics are:

Purpose

Provides a lock for the debug registers. The OS Lock also disables some debug exceptions and debug events.

Configuration

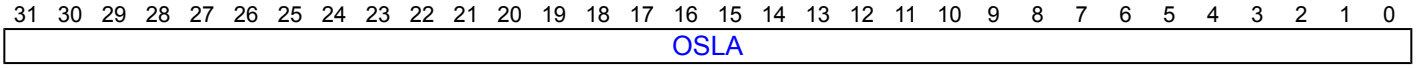
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGOSLAR are UNDEFINED.

The OS Lock can also be locked or unlocked using the AArch64 System register OSLAR_EL1 and External register OSLAR_EL1.

Attributes

DBGOSLAR is a 32-bit register.

Field descriptions



OSLA, bits [31:0]

OS Lock Access. Writing the value `0xC5ACCE55` to the DBGOSLAR sets the OS Lock to 1. Writing any other value sets the OS Lock to 0.

Use DBGOSLSR.OSLK to check the current status of the lock.

Accessing DBGOSLAR

Accesses to this register use the following encodings in the System register encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0001	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDOSA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGOSLAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGOSLAR = R[t];
elsif PSTATE.EL == EL3 then
    DBGOSLAR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGOSLSR, Debug OS Lock Status Register

The DBGOSLSR characteristics are:

Purpose

Provides status information for the OS Lock.

Configuration

AArch32 System register DBGOSLSR bits [31:0] are architecturally mapped to AArch64 System register OSLSR_EL1[31:0].

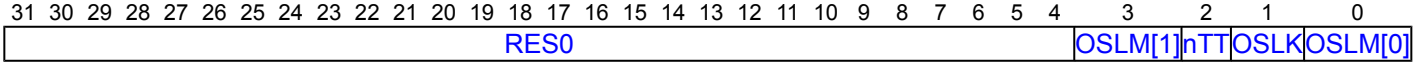
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGOSLSR are UNDEFINED.

The OS Lock status is also visible in the external debug interface through EDPRSR.

Attributes

DBGOSLSR is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

OSLM, bits [3, 0]

OS Lock model implemented. Identifies the form of OS save and restore mechanism implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OSLM	Meaning
0b00	OS Lock not implemented.
0b10	OS Lock implemented.

All other values are reserved. In an Armv8 implementation the value 0b00 is not permitted.

The OSLM field is split as follows:

- OSLM[1] is DBGOSLSR[3].
- OSLM[0] is DBGOSLSR[0].

Access to this field is **RO**.

nTT, bit [2]

Not 32-bit access. This bit is always RAZ. It indicates that a 32-bit access is needed to write the key to the OS Lock Access Register.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

OSLK, bit [1]

OS Lock Status.

OSLK	Meaning
0b0	OS Lock unlocked.
0b1	OS Lock locked.

The OS Lock is locked and unlocked by writing to the OS Lock Access Register.

The reset behavior of this field is:

- On a Cold reset, this field resets to '1'.

Accessing DBGOSLSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0001	0b0001	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDOSA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGOSLSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGOSLSR;
elsif PSTATE.EL == EL3 then
    R[t] = DBGOSLSR;
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGPRCR, Debug Power Control Register

The DBGPRCR characteristics are:

Purpose

Controls behavior of the PE on powerdown request.

Configuration

AArch32 System register DBGPRCR bits [31:0] are architecturally mapped to AArch64 System register DBGPRCR_EL1[31:0].

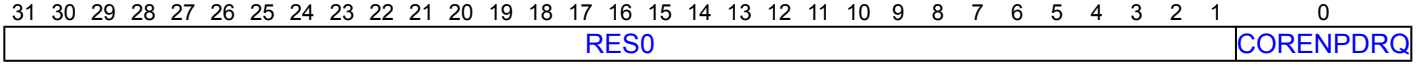
AArch32 System register DBGPRCR bit [0] is architecturally mapped to External register EDPRCR[0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGPRCR are UNDEFINED.

Attributes

DBGPRCR is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

CORENPDRQ, bit [0]

When FEAT_DoPD is implemented:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

CORENPDRQ	Meaning
0b0	If the system responds to a powerdown request, it powers down Core power domain.
0b1	If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGNOPWRDWN** signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the Cold reset value on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

On a Cold reset, if the powerup request is implemented and the powerup request has been asserted, this field is set to an IMPLEMENTATION DEFINED choice of 0 or 1. If the powerup request is not asserted, this field is set to 0.

Otherwise:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

CORENPDRQ	Meaning
0b0	If the system responds to a powerdown request, it powers down Core power domain.
0b1	If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGNOPWRDWN** signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the value of EDPRCR.COREPURQ on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The reset behavior of this field is:

- On a Cold reset, this field resets to the value in EDPRCR.COREPURQ.

Accessing DBGPRCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0001	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDOSA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGPRCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGPRCR;
elsif PSTATE.EL == EL3 then
    R[t] = DBGPRCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0001	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDOSA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGPRCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGPRCR = R[t];
elsif PSTATE.EL == EL3 then
    DBGPRCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGVCR, Debug Vector Catch Register

The DBGVCR characteristics are:

Purpose

Controls Vector Catch debug events.

Configuration

AArch32 System register DBGVCR bits [31:0] are architecturally mapped to AArch64 System register DBGVCR32_EL2[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGVCR are UNDEFINED.

This register is required in all implementations.

Attributes

DBGVCR is a 32-bit register.

Field descriptions

When EL3 is implemented and EL3 is using AArch32:

31	30	29	28	27	26	25	24 23 22 21 20 19 18 17 16	15	14	13	12	11	10	9 8	7	6	5	4	3	2	1	0
NSF	NSI	RES0	NSD	NSP	NSS	NSU	RES0	MF	MI	RES0	MD	MP	MS	RES0	SF	SI	RES0	SD	SP	SS	SU	RES0

NSF, bit [31]

FIQ vector catch enable in Non-secure state.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSI, bit [30]

IRQ vector catch enable in Non-secure state.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

NSD, bit [28]

Data Abort exception vector catch enable in Non-secure state.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSP, bit [27]

Prefetch Abort vector catch enable in Non-secure state.

The exception vector offset is `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSS, bit [26]

Supervisor Call (SVC) vector catch enable in Non-secure state.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSU, bit [25]

Undefined Instruction vector catch enable in Non-secure state.

The exception vector offset is `0x04`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [24:16]

Reserved, RES0.

MF, bit [15]

FIQ vector catch enable in Monitor mode.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MI, bit [14]

IRQ vector catch enable in Monitor mode.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

MD, bit [12]

Data Abort exception vector catch enable in Monitor mode.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MP, bit [11]

Prefetch Abort vector catch enable in Monitor mode.

The exception vector offset is `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MS, bit [10]

Secure Monitor Call (SMC) vector catch enable in Monitor mode.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:8]

Reserved, RES0.

SF, bit [7]

FIQ vector catch enable in Secure state.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SI, bit [6]

IRQ vector catch enable in Secure state.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

SD, bit [4]

Data Abort exception vector catch enable in Secure state.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SP, bit [3]

Prefetch Abort vector catch enable in Secure state.

The exception vector offset is `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SS, bit [2]

Supervisor Call (SVC) vector catch enable in Secure state.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SU, bit [1]

Undefined Instruction vector catch enable in Secure state.

The exception vector offset is `0x04`.

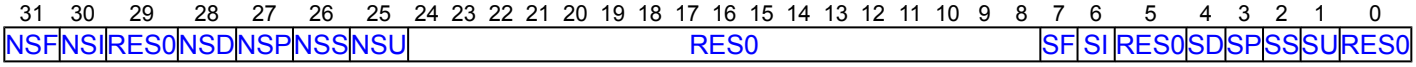
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

When EL3 is implemented and EL3 is using AArch64:



NSF, bit [31]

FIQ vector catch enable in Non-secure state.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSI, bit [30]

IRQ vector catch enable in Non-secure state.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

NSD, bit [28]

Data Abort exception vector catch enable in Non-secure state.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSP, bit [27]

Prefetch Abort vector catch enable in Non-secure state.

The exception vector offset is `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSS, bit [26]

Supervisor Call (SVC) vector catch enable in Non-secure state.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSU, bit [25]

Undefined Instruction vector catch enable in Non-secure state.

The exception vector offset is `0x04`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [24:8]

Reserved, RES0.

SF, bit [7]

FIQ vector catch enable in Secure state.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SI, bit [6]

IRQ vector catch enable in Secure state.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

SD, bit [4]

Data Abort exception vector catch enable in Secure state.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SP, bit [3]

Prefetch Abort vector catch enable in Secure state.

The exception vector offset is `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SS, bit [2]

Supervisor Call (SVC) vector catch enable in Secure state.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SU, bit [1]

Undefined Instruction vector catch enable in Secure state.

The exception vector offset is `0x04`.

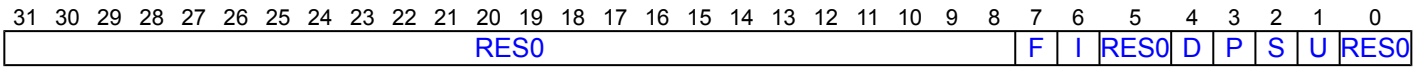
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

When EL3 is not implemented:



Bits [31:8]

Reserved, RES0.

F, bit [7]

FIQ vector catch enable.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [6]

IRQ vector catch enable.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

D, bit [4]

Data Abort exception vector catch enable.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P, bit [3]

Prefetch Abort vector catch enable.

The exception vector offset `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [2]

Supervisor Call (SVC) vector catch enable.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

U, bit [1]

Undefined Instruction vector catch enable.

The exception vector offset is `0x04`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

Accessing DBGVCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGVCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGVCR;
elsif PSTATE.EL == EL3 then
    R[t] = DBGVCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGVCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGVCR = R[t];
elsif PSTATE.EL == EL3 then
    DBGVCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWCR<n>, Debug Watchpoint Control Registers, n = 0 - 15

The DBGWCR<n> characteristics are:

Purpose

Holds control information for a watchpoint. Forms watchpoint n together with value register DBGWVR<n>.

Configuration

AArch32 System register DBGWCR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGWCR<n>_EL1[31:0].

AArch32 System register DBGWCR<n> bits [31:0] are architecturally mapped to External register DBGWCR<n>_EL1[31:0].

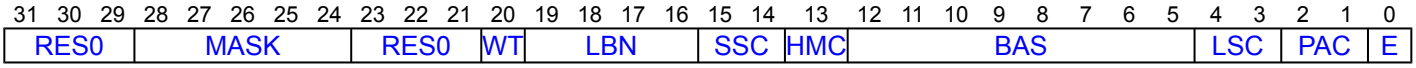
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGWCR<n> are UNDEFINED.

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGWCR<n> is a 32-bit register.

Field descriptions



When the E field is zero, all the other fields in the register are ignored.

Bits [31:29]

Reserved, RES0.

MASK, bits [28:24]

Address Mask. Only objects up to 2GB can be watched using a single mask.

MASK	Meaning
0b00000	No mask.
0b00011..0b11111	Number of address bits masked.

All other values are reserved.

Indicates the number of masked address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b11111 masking 31 address bits (0x7FFFFFFF mask for address).

If programmed with a reserved value, the watchpoint behaves as if either:

- DBGWCR<n>.MASK has been programmed with a defined value, which might be 0 (no mask), other than for a direct read of DBGWCR<n>.
- The watchpoint is disabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [23:21]

Reserved, RES0.

WT, bit [20]

Watchpoint type. Possible values are:

WT	Meaning
0b0	Unlinked data address match.
0b1	Linked data address match.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked Breakpoint Number. For Linked data address watchpoints, this specifies the index of the Context-matching breakpoint linked to.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the HMC and PAC fields.

For more information, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions', and 'Reserved DBGWCR<n>.{SSC, HMC, PAC} values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

BAS, bits [12:5]

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by DBGWVR<n> is being watched.

BAS	Description
0bxxxxxxx1	Match byte at DBGWVR<n>
0bxxxxxx1x	Match byte at DBGWVR<n>+1
0bxxxxx1xx	Match byte at DBGWVR<n>+2
0bxxxx1xxx	Match byte at DBGWVR<n>+3

In cases where DBGWVR<n> addresses a double-word:

BAS	Description, if DBGWVR<n>[2] == 0
0bxxx1xxxx	Match byte at DBGWVR<n>+4
0bxx1xxxxx	Match byte at DBGWVR<n>+5
0bx1xxxxxx	Match byte at DBGWVR<n>+6
0b1xxxxxxx	Match byte at DBGWVR<n>+7

If DBGWVR<n>[2] == 1, only BAS[3:0] are used and BAS[7:4] are ignored. Arm deprecates setting DBGWVR<n>[2] == 1.

The valid values for BAS are nonzero binary numbers all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<n>.BAS values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LSC, bits [4:3]

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

LSC	Meaning
0b01	Match instructions that load from a watchpointed address.
0b10	Match instructions that store to a watchpointed address.
0b11	Match instructions that load from or store to a watchpointed address.

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PAC, bits [2:1]

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and HMC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable watchpoint n. Possible values are:

E	Meaning
0b0	Watchpoint disabled.
0b1	Watchpoint enabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGWCR<n>

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	m[3:0]	0b111

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif m >= NUM_WATCHPOINTS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGWCR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGWCR[m];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGWCR[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	m[3:0]	0b111

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif m >= NUM_WATCHPOINTS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR[m] = R[t];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWFAR, Debug Watchpoint Fault Address Register

The DBGWFAR characteristics are:

Purpose

Previously returned information about the address of the instruction that accessed a watchpointed address. Is now deprecated and RES0.

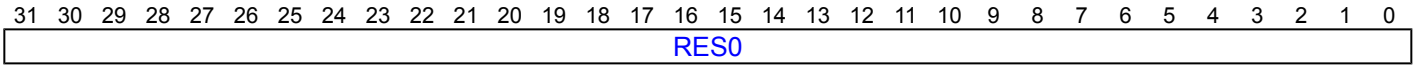
Configuration

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGWFAR are UNDEFINED.

Attributes

DBGWFAR is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing DBGWFAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGWFAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        R[t] = DBGWFAR;
elsif PSTATE.EL == EL3 then
    R[t] = DBGWFAR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGWFAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGWFAR = R[t];
elsif PSTATE.EL == EL3 then
    DBGWFAR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWVR<n>, Debug Watchpoint Value Registers, n = 0 - 15

The DBGWVR<n> characteristics are:

Purpose

Holds a data address value for use in watchpoint matching. Forms watchpoint n together with control register DBGWCR<n>.

Configuration

AArch32 System register DBGWVR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGWVR<n>_EL1[31:0].

AArch32 System register DBGWVR<n> bits [31:0] are architecturally mapped to External register DBGWVR<n>_EL1[31:0].

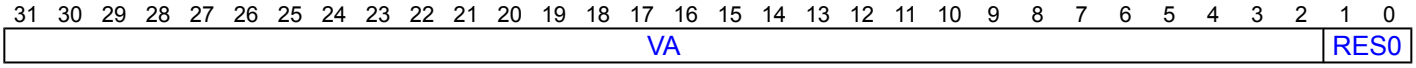
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DBGWVR<n> are UNDEFINED.

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGWVR<n> is a 32-bit register.

Field descriptions



VA, bits [31:2]

Bits[31:2] of the address value for comparison.

Arm deprecates setting DBGWVR<n>[2] == 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

Accessing DBGWVR<n>

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	m[3:0]	0b110

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif m >= NUM_WATCHPOINTS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGWVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGWVR[m];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        R[t] = DBGWVR[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b000	0b0000	m[3:0]	0b110

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif m >= NUM_WATCHPOINTS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWVR[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWVR[m] = R[t];
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWVR[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCCIMVAC, Data Cache line Clean and Invalidate by VA to PoC

The DCCIMVAC characteristics are:

Purpose

Clean and Invalidate data or unified cache line by virtual address to PoC.

Configuration

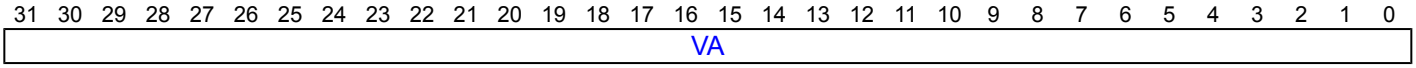
AArch32 System instruction DCCIMVAC performs the same function as AArch64 System instruction DC CIVAC.

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DCCIMVAC are UNDEFINED.

Attributes

DCCIMVAC is a 32-bit System instruction.

Field descriptions



VA, bits [31:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DCCIMVAC

Execution of this instruction might require an address translation from VA to PA, and that translation might fault.

For more information about faults, see 'Permission fault'.

For more information about data cache maintenance instructions, see 'AArch32 data cache maintenance instructions (DC*)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1110	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TPCP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TPC ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCCISW, Data Cache line Clean and Invalidate by Set/Way

The DCCISW characteristics are:

Purpose

Clean and Invalidate data or unified cache line by set/way.

Configuration

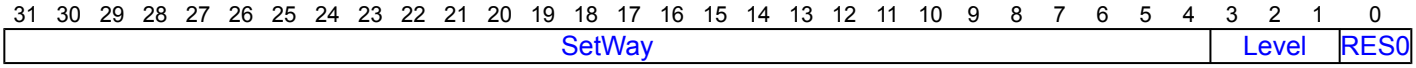
AArch32 System instruction DCCISW performs the same function as AArch64 System instruction DC CISW.

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DCCISW are UNDEFINED.

Attributes

DCCISW is a 32-bit System instruction.

Field descriptions



SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DCCISW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1110	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TSW == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TSW ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCCMVAC, Data Cache line Clean by VA to PoC

The DCCMVAC characteristics are:

Purpose

Clean data or unified cache line by virtual address to PoC.

Configuration

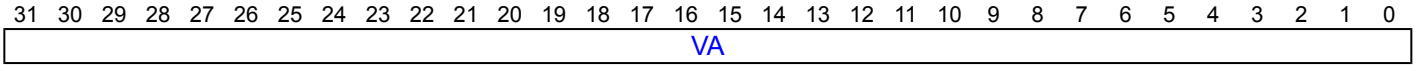
AArch32 System instruction DCCMVAC performs the same function as AArch64 System instruction DC CVAC.

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DCCMVAC are UNDEFINED.

Attributes

DCCMVAC is a 32-bit System instruction.

Field descriptions



VA, bits [31:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DCCMVAC

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'AArch32 data cache maintenance instructions (DC*)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1010	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TPCP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TPC ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCCMVAU, Data Cache line Clean by VA to PoU

The DCCMVAU characteristics are:

Purpose

Clean data or unified cache line by virtual address to PoU.

Configuration

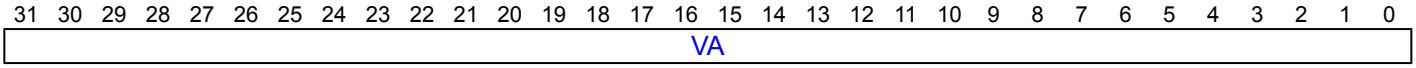
AArch32 System instruction DCCMVAU performs the same function as AArch64 System instruction DC CVAU.

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DCCMVAU are UNDEFINED.

Attributes

DCCMVAU is a 32-bit System instruction.

Field descriptions



VA, bits [31:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DCCMVAU

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'AArch32 data cache maintenance instructions (DC*)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1011	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TOCU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TPU ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TOCU
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoU);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoU);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoU);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCCSW, Data Cache line Clean by Set/Way

The DCCSW characteristics are:

Purpose

Clean data or unified cache line by set/way.

Configuration

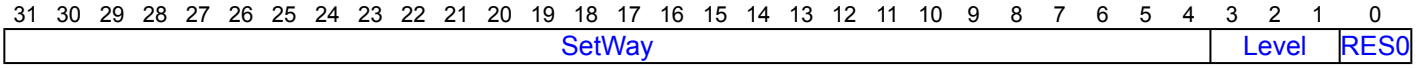
AArch32 System instruction DCCSW performs the same function as AArch64 System instruction DC CSW.

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DCCSW are UNDEFINED.

Attributes

DCCSW is a 32-bit System instruction.

Field descriptions



SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DCCSW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b1010	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TSW == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TSW ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCIMVAC, Data Cache line Invalidate by VA to PoC

The DCIMVAC characteristics are:

Purpose

Invalidate data or unified cache line by virtual address to PoC.

Configuration

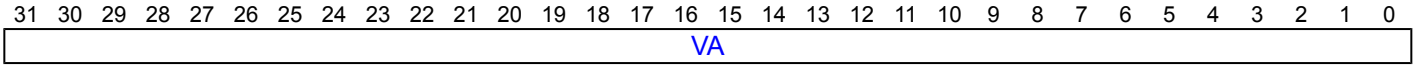
AArch32 System instruction DCIMVAC performs the same function as AArch64 System instruction DC IVAC.

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DCIMVAC are UNDEFINED.

Attributes

DCIMVAC is a 32-bit System instruction.

Field descriptions



VA, bits [31:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing DCIMVAC

It is IMPLEMENTATION DEFINED whether, when this instruction is executed, it can generate a watchpoint. If this instruction can generate a watchpoint this is prioritized in the same way as other watchpoints.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'AArch32 data cache maintenance instructions (DC*)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0110	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TPCP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TPC ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_PoC);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCISW, Data Cache line Invalidate by Set/Way

The DCISW characteristics are:

Purpose

Invalidate data or unified cache line by set/way.

Configuration

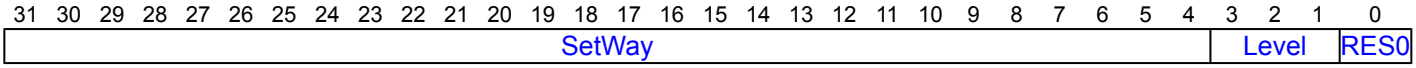
AArch32 System instruction DCISW performs the same function as AArch64 System instruction DC ISW.

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DCISW are UNDEFINED.

Attributes

DCISW is a 32-bit System instruction.

Field descriptions



SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log_2(ASSOCIATIVITY), L = Log_2(LINELEN), B = (L + S), S = Log_2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing DCISW

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0110	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TSW == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TSW ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_SetWay);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DFAR, Data Fault Address Register

The DFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Data Abort exception.

Configuration

This register is banked between DFAR and DFAR_S and DFAR_NS.

AArch32 System register DFAR bits [31:0] are architecturally mapped to AArch64 System register FAR_EL1[31:0].

AArch32 System register DFAR bits [31:0] (DFAR_S) are architecturally mapped to AArch32 System register HDFAR[31:0] when FEAT_AA32EL2 is implemented and FEAT_AA32EL3 is implemented.

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DFAR are UNDEFINED.

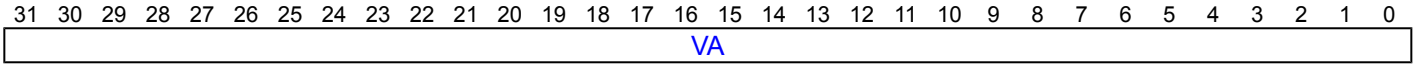
Attributes

DFAR is a 32-bit register.

This register has the following instances:

- DFAR, when EL3 is not implemented or FEAT_AA64 is implemented.
- DFAR_S, when FEAT_AA32EL3 is implemented.
- DFAR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



VA, bits [31:0]

VA of faulting address of synchronous Data Abort exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DFAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T6
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T6 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = DFAR_NS;
    else
        R[t] = DFAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = DFAR_NS;
    else
        R[t] = DFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = DFAR_S;
    else
        R[t] = DFAR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T6
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T6 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        DFAR_NS = R[t];
    else
        DFAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        DFAR_NS = R[t];
    else
        DFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        DFAR_S = R[t];
    else
        DFAR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DFSR, Data Fault Status Register

The DFSR characteristics are:

Purpose

Holds status information about the last data fault.

Configuration

This register is banked between DFSR and DFSR_S and DFSR_NS.

AArch32 System register DFSR bits [31:0] are architecturally mapped to AArch64 System register ESR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DFSR are UNDEFINED.

The current translation table format determines which format of the register is used.

Attributes

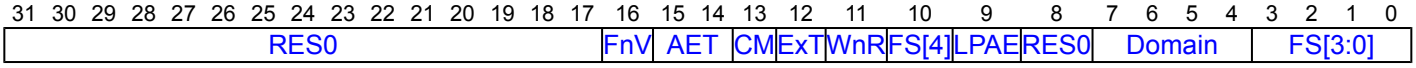
DFSR is a 32-bit register.

This register has the following instances:

- DFSR, when EL3 is not implemented or FEAT_AA64 is implemented.
- DFSR_S, when FEAT_AA32EL3 is implemented.
- DFSR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When TTBCR.EAE == 0:



Bits [31:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	DFAR is valid.
0b1	DFAR is not valid, and holds an UNKNOWN value.

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Data Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AET, bits [15:14]

When FEAT_RAS is implemented:

Asynchronous Error Type. When DFSC is `0b010001`, describes the PE error state after taking the SError exception. Possible values are:

AET	Meaning
`0b00`	Uncontainable (UC).
`0b01`	Unrecoverable state (UEU).
`0b10`	Restartable state (UEO).
`0b11`	Recoverable state (UER).

This field is valid only if the DFSC code is `0b010001`. It is RES0 for all other aborts.

In the event of multiple errors taken as a single SError exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CM, bit [13]

Cache maintenance fault. For synchronous faults, this bit indicates whether a cache maintenance instruction generated the fault.

CM	Meaning
`0b0`	Abort not caused by execution of a cache maintenance instruction.
`0b1`	Abort caused by execution of a cache maintenance instruction, or on an address translation.

On a synchronous Data Abort exception on a translation table walk, this bit is UNKNOWN.

On an asynchronous fault, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ExT, bit [12]

External abort type.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [11]

Write not Read bit. Indicates whether the abort was caused by a write or a read instruction.

WnR	Meaning
0b0	Abort caused by a read instruction.
0b1	Abort caused by a write instruction.

For faults on the cache maintenance and address translation System instructions in the (coproc==0b1111) encoding space this bit always returns a value of 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FS, bits [10, 3:0]

Fault status bits. Possible values of FS[4:0] are:

FS	Meaning	Applies when
0b00001	Alignment fault.	
0b00010	Debug exception.	
0b00011	Access flag fault, level 1.	
0b00100	Fault on instruction cache maintenance.	
0b00101	Translation fault, level 1.	
0b00110	Access flag fault, level 2.	
0b00111	Translation fault, level 2.	
0b01000	Synchronous External abort, not on translation table walk.	
0b01001	Domain fault, level 1.	
0b01011	Domain fault, level 2.	
0b01100	Synchronous External abort, on translation table walk, level 1.	
0b01101	Permission fault, level 1.	
0b01110	Synchronous External abort, on translation table walk, level 2.	
0b01111	Permission fault, level 2.	
0b10000	TLB conflict abort.	
0b10100	IMPLEMENTATION DEFINED fault (Lockdown fault).	
0b10101	IMPLEMENTATION DEFINED fault (Unsupported Exclusive access fault).	
0b10110	SError exception.	
0b11000	SError exception, from a parity or ECC error on memory access.	When FEAT_RAS is not implemented
0b11001	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b11100	Synchronous parity or ECC error on translation table walk, level 1.	When FEAT_RAS is not implemented
0b11110	Synchronous parity or ECC error on translation table walk, level 2.	When FEAT_RAS is not implemented

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Short-descriptor translation table lookup'.

The FS field is split as follows:

- FS[4] is DFSR[10].
- FS[3:0] is DFSR[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

LPAE	Meaning
0b0	Using the Short-descriptor translation table formats.
0b1	Using the Long-descriptor translation table formats.

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

Domain, bits [7:4]

The domain of the fault address.

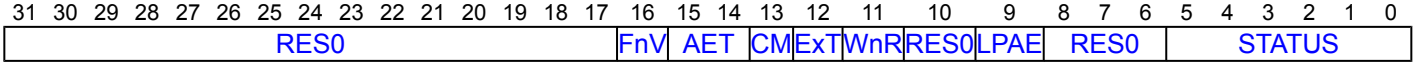
Arm deprecates any use of this field, see 'The Domain field in the DFSR'.

This field is UNKNOWN for certain faults where the DFSR is updated and reported using the Short-descriptor FSR encodings, see 'Validity of Domain field on faults that update the DFSR when using the Short-descriptor encodings'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When TTBCR.EAE == 1:



Bits [31:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	DFAR is valid.
0b1	DFAR is not valid, and holds an UNKNOWN value.

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Data Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AET, bits [15:14]

When FEAT_RAS is implemented:

Asynchronous Error Type. When DFSC is 0b010001, describes the PE error state after taking the SError exception. Possible values are:

AET	Meaning
0b00	Uncontainable (UC).
0b01	Unrecoverable state (UEU).
0b10	Restartable state (UEO).
0b11	Recoverable state (UER).

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other aborts.

In the event of multiple errors taken as a single SError exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CM, bit [13]

Cache maintenance fault. For synchronous faults, this bit indicates whether a cache maintenance instruction generated the fault.

CM	Meaning
0b0	Abort not caused by execution of a cache maintenance instruction.
0b1	Abort caused by execution of a cache maintenance instruction.

On a synchronous Data Abort exception on a translation table walk, this bit is UNKNOWN.

On an asynchronous fault, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ExT, bit [12]

External abort type.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [11]

Write not Read bit. Indicates whether the abort was caused by a write or a read instruction.

WnR	Meaning
0b0	Abort caused by a read instruction.
0b1	Abort caused by a write instruction.

For faults on the cache maintenance and address translation System instructions in the (coproc==0b1111) encoding space this bit always returns a value of 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [10]

Reserved, RES0.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

LPAE	Meaning
0b0	Using the Short-descriptor translation table formats.
0b1	Using the Long-descriptor translation table formats.

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status bits. Possible values of this field are:

STATUS	Meaning	Applies when
0b000000	Address size fault in translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk.	
0b010001	Asynchronous SError exception.	
0b010101	Synchronous External abort on translation table walk, level 1.	
0b010110	Synchronous External abort on translation table walk, level 2.	
0b010111	Synchronous External abort on translation table walk, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011001	Asynchronous SError exception, from a parity or ECC error on memory access.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk, level 3.	When FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Debug exception.	
0b110000	TLB conflict abort.	
0b110100	IMPLEMENTATION DEFINED fault (Lockdown).	
0b110101	IMPLEMENTATION DEFINED fault (Unsupported Exclusive access).	

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DFSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = DFSR_NS;
    else
        R[t] = DFSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = DFSR_NS;
    else
        R[t] = DFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = DFSR_S;
    else
        R[t] = DFSR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        DFSR_NS = R[t];
    else
        DFSR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        DFSR_NS = R[t];
    else
        DFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        DFSR_S = R[t];
    else
        DFSR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DISR, Deferred Interrupt Status Register

The DISR characteristics are:

Purpose

Records that an SError exception has been consumed by an `ESB` instruction.

Configuration

AArch32 System register DISR bits [31:0] are architecturally mapped to AArch64 System register DISR_EL1[31:0].

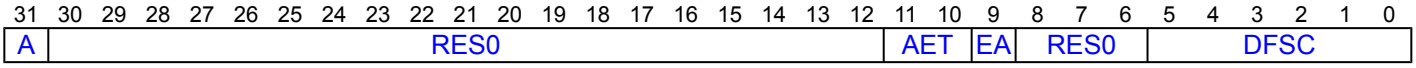
This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DISR are UNDEFINED.

Attributes

DISR is a 32-bit register.

Field descriptions

When the ESB instruction is executed at EL2:



A, bit [31]

Set to 1 when an `ESB` instruction defers an asynchronous SError exception. If the implementation does not include any sources of SError exception that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:12]

Reserved, RES0.

AET, bits [11:10]

Asynchronous Error Type. See the description of HSR.AET for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort Type. See the description of HSR.EA for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

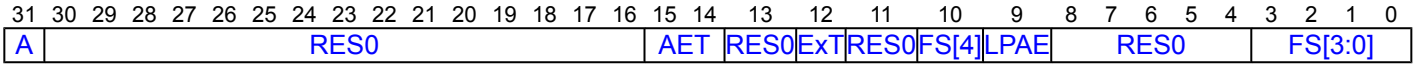
DFSC, bits [5:0]

Fault Status Code. See the description of HSR.DFSC for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When the ESB instruction is executed at EL0 or EL1 and where TTBCR.EAE == 0:



A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError exception. If the implementation does not include any sources of SError exception that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

Asynchronous Error Type. See the description of DFSR.AET for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

External abort Type. See the description of DFSR.ExT for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault Status Code. See the description of DFSR.FS for an SError exception.

The FS field is split as follows:

- FS[4] is DISR[10].

- FS[3:0] is DISR[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

Format.

LPAE	Meaning
0b0	Using the Short-descriptor translation table format.

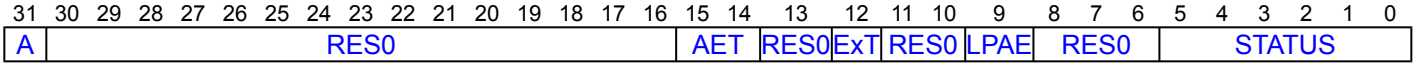
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When the ESB instruction is executed at EL0 or EL1 and where TTBCR.EAE == 1:



A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError exception. If the implementation does not include any sources of SError exception that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

Asynchronous Error Type. See the description of DFSR.AET for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

External abort Type. See the description of DFSR.ExT for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

Format.

LPAE	Meaning
0b1	Using the Long-descriptor translation table format.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault Status Code. See the description of DFSR.FS for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DISR

An indirect write to DISR made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR occurring in program order after the ESB instruction.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
(HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) && IsHCRXEL2Enabled() &&
HCRX_EL2.TMEA == '1')) then
        R[t] = VDISR_EL2<31:0>;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.AMO ==
'1' then
        R[t] = VDISR;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && !Halted()
&& SCR_EL3.EA == '1' then
        R[t] = Zeros(32);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && !Halted() &&
SCR.EA == '1' then
        R[t] = Zeros(32);
    else
        R[t] = DISR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && !Halted() &&
SCR_EL3.EA == '1' then
        R[t] = Zeros(32);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && !Halted() &&
SCR.EA == '1' then
        R[t] = Zeros(32);
    else
        R[t] = DISR;
elsif PSTATE.EL == EL3 then
    R[t] = DISR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
(HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) && IsHCRXEL2Enabled() &&
HCRX_EL2.TMEA == '1')) then
        VDISR_EL2 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.AMO ==
'1' then
        VDISR = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && !Halted()
&& SCR_EL3.EA == '1' then
        return;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && !Halted() &&
SCR.EA == '1' then
        return;
    else
        DISR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && !Halted() &&
SCR_EL3.EA == '1' then
        return;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && !Halted() &&
SCR.EA == '1' then
        return;
    else
        DISR = R[t];
elsif PSTATE.EL == EL3 then
    DISR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DLR, Debug Link Register

The DLR characteristics are:

Purpose

In Debug state, holds the address to restart from.

Configuration

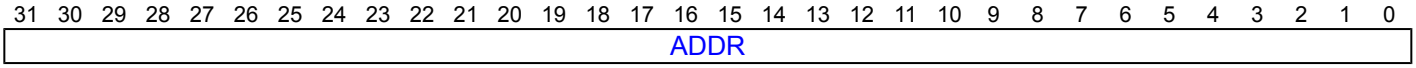
AArch32 System register DLR bits [31:0] are architecturally mapped to AArch64 System register DLR_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to DLR are UNDEFINED.

Attributes

DLR is a 32-bit register.

Field descriptions



ADDR, bits [31:0]

Restart address.

Accessing DLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b011	0b0100	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    R[t] = DLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b011	0b0100	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    DLR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DSPSR, Debug Saved Program Status Register

The DSPSR characteristics are:

Purpose

Holds the saved process state for Debug state. On entering Debug state, PSTATE information is written to this register. On exiting Debug state, values are copied from this register to PSTATE.

Configuration

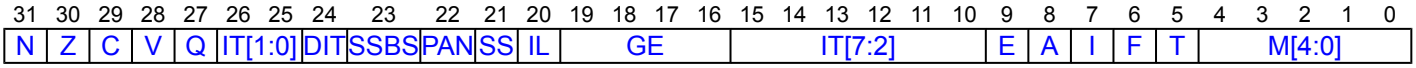
AArch32 System register DSPSR bits [31:0] are architecturally mapped to AArch64 System register DSPSR_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to DSPSR are UNDEFINED.

Attributes

DSPSR is a 32-bit register.

Field descriptions



N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on entering Debug state, and copied to PSTATE.N on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on entering Debug state, and copied to PSTATE.Z on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on entering Debug state, and copied to PSTATE.C on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on entering Debug state, and copied to PSTATE.V on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on entering Debug state, and copied to PSTATE.Q on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on entering Debug state, and copied to PSTATE.IT on exiting Debug state.

On exiting Debug state, DSPSR.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is DSPSR[26:25].
- IT[7:2] is DSPSR[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on entering Debug state, and copied to PSTATE.DIT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on entering Debug state, and copied to PSTATE.SSBS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on entering Debug state, and copied to PSTATE.PAN on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on entering Debug state, and conditionally copied to PSTATE.SS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on entering Debug state, and copied to PSTATE.IL on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on entering Debug state, and copied to PSTATE.GE on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on entering Debug state, and copied to PSTATE.E on exiting Debug state.

If the implementation does not support big-endian operation, DSPSR.E is RES0. If the implementation does not support little-endian operation, DSPSR.E is RES1. On exiting Debug state, if the implementation does not support big-endian operation at the Exception level being returned to, DSPSR.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, DSPSR.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on entering Debug state, and copied to PSTATE.A on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on entering Debug state, and copied to PSTATE.I on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on entering Debug state, and copied to PSTATE.F on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on entering Debug state, and copied to PSTATE.T on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on entering Debug state, and copied to PSTATE.M[4:0] on exiting Debug state.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10110	Monitor.
0b10111	Abort.
0b11010	Hyp.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If DSPSR.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DSPSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b011	0b0100	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    R[t] = DSPSR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b011	0b0100	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    DSPSR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DSPSR2, Debug Saved Process State Register 2

The DSPSR2 characteristics are:

Purpose

Holds the saved process state for Debug state. On entering Debug state, PSTATE information is written to this register. On exiting Debug state, values are copied from this register to PSTATE.

Configuration

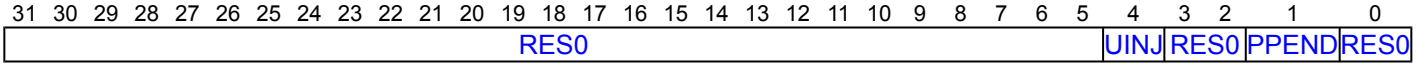
AArch32 System register DSPSR2 bits [31:0] are architecturally mapped to AArch64 System register DSPSR_EL0[63:32] when FEAT_Debugv8p9 is implemented.

This register is present only when FEAT_Debugv8p9 is implemented and FEAT_AA32 is implemented. Otherwise, direct accesses to DSPSR2 are UNDEFINED.

Attributes

DSPSR2 is a 32-bit register.

Field descriptions



Bits [31:5]

Reserved, RES0.

UINJ, bit [4]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Set to the value of PSTATE.UINJ on entering Debug state, and copied to PSTATE.UINJ on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [3:2]

Reserved, RES0.

PPEND, bit [1]

When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Set to the value of PSTATE.PPEND on entering Debug state, and conditionally copied to PSTATE.PPEND on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [0]

Reserved, RES0.

Accessing DSPSR2

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b011	0b0100	0b0101	0b010

```
if !(IsFeatureImplemented(FEAT_Debugv8p9) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    R[t] = DSPSR2;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b011	0b0100	0b0101	0b010

```
if !(IsFeatureImplemented(FEAT_Debugv8p9) && IsFeatureImplemented(FEAT_AA32)) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    DSPSR2 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DTLBIALL, Data TLB Invalidate All

The DTLBIALL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from data TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
  - Would be required for the EL1&0 translation regime.
  - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2, and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the Non-secure PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DTLBIALL are UNDEFINED.

Attributes

DTLBIALL is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing DTLBIALL

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.DTLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_ExcludeXS);
        else
            AArch32.DTLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_AllAttr);
elsif PSTATE.EL == EL2 then
    AArch32.DTLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    AArch32.DTLBI_ALL(SecurityStateAtEL(EL3), Regime_EL30, Broadcast_NSH, TLBI_AllAttr);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DTLBIASID, Data TLB Invalidate by ASID match

The DTLBIASID characteristics are:

Purpose

Invalidate all cached copies of translation table entries from data TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

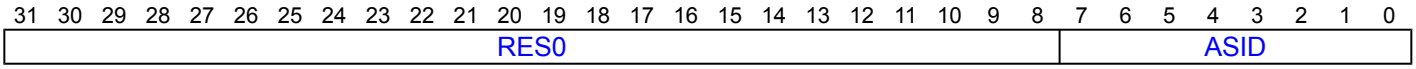
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DTLBIASID are UNDEFINED.

Attributes

DTLBIASID is a 32-bit System instruction.

Field descriptions



Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

Executing DTLBIASID

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0110	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.DTLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, R[t]);
        else
            AArch32.DTLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.DTLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBI_AllAttr,
R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.DTLBI_ASID(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_NSH,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DTLBIMVA, Data TLB Invalidate by VA

The DTLBIMVA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from data TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

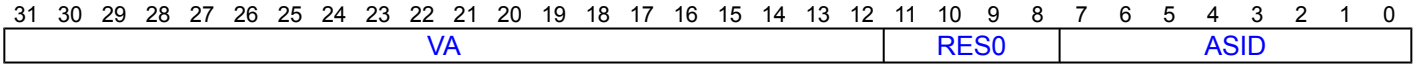
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to DTLBIMVA are UNDEFINED.

Attributes

DTLBIMVA is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

Executing DTLBIMVA

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0110	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.DTLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.DTLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.DTLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.DTLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DVPRCTX, Data Value Prediction Restriction by Context

The DVPRCTX characteristics are:

Purpose

Data Value Prediction Restriction by Context applies to all Data Value Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

Note

The prediction of the PSTATE.{N,Z,C,V} values is not considered a data value for this purpose.

Data value predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control speculative execution occurring after the instruction is complete and synchronized.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

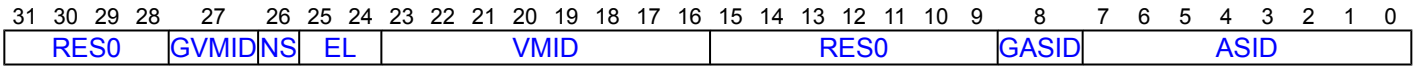
Configuration

This instruction is present only when FEAT_AA32 is implemented and FEAT_SPECRES is implemented. Otherwise, direct accesses to DVPRCTX are UNDEFINED.

Attributes

DVPRCTX is a 32-bit System instruction.

Field descriptions



Bits [31:28]

Reserved, RES0.

GVMID, bit [27]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
0b0	Applies to specified VMID for an EL0 or EL1 target execution context.
0b1	Applies to all VMIDs for an EL0 or EL1 target execution context.

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

NS, bit [26]

Security State.

NS	Meaning
0b0	Secure state.
0b1	Non-secure state.

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

EL	Meaning
0b00	EL0.
0b01	EL1.
0b10	EL2.
0b11	EL3.

If the instruction is executed at an Exception level lower than the specified level, or is specified to apply to a combination of Exception level and Security state that is not implemented, this instruction is treated as a NOP.

VMID, bits [23:16]

Only applies when bit[27] is 0 and the target execution context is either:

- EL1.
- EL0 when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1} or EL2 is using AArch32.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and (the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1} or ELUsingAArch32(EL2)), this field is treated as the current VMID.

When the instruction is executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

Bits [15:9]

Reserved, RES0.

GASID, bit [8]

Execution of this instruction applies to all ASIDs or a specified ASID.

GASID	Meaning
0b0	Applies to specified ASID for an EL0 target execution context.
0b1	Applies to all ASIDs for an EL0 target execution context.

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [7:0]

Only applies for an EL0 target execution context and when bit[8] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

Executing DVPRCTX

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0011	0b101

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_SPECRES)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && SCTLR.EnRCTX == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T7
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DVPRCTX == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnRCTX == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch32.RestrictPrediction(R[t], RestrictType_DataValue);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x03);
    else
        AArch32.RestrictPrediction(R[t], RestrictType_DataValue);
elsif PSTATE.EL == EL2 then
    AArch32.RestrictPrediction(R[t], RestrictType_DataValue);
elsif PSTATE.EL == EL3 then
    AArch32.RestrictPrediction(R[t], RestrictType_DataValue);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ELR_hyp, Exception Link Register (Hyp mode)

The ELR_hyp characteristics are:

Purpose

When taking an exception to Hyp mode, holds the address to return to.

Configuration

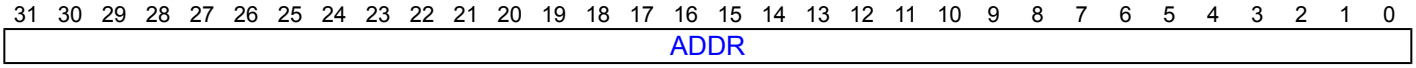
AArch32 System register ELR_hyp bits [31:0] are architecturally mapped to AArch64 System register ELR_EL2[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to ELR_hyp are UNDEFINED.

Attributes

ELR_hyp is a 32-bit register.

Field descriptions



ADDR, bits [31:0]

Return address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ELR_hyp

ELR_hyp is accessible only at Hyp mode and Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS{<c>}{<q>} <Rd>, ELR_hyp
```

R	M	M1
0b0	0b1	0b1110

```
MSR{<c>}{<q>} ELR_hyp, <Rn>
```

R	M	M1
0b0	0b1	0b1110

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRIDR, Error Record ID Register

The ERRIDR characteristics are:

Purpose

Defines the highest numbered index of the error records that can be accessed through the Error Record System registers.

Configuration

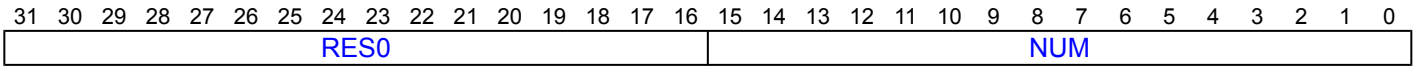
AArch32 System register ERRIDR bits [31:0] are architecturally mapped to AArch64 System register ERRIDR_EL1[31:0].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERRIDR are UNDEFINED.

Attributes

ERRIDR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

NUM, bits [15:0]

Highest numbered index of the records that can be accessed through the Error Record System registers plus one. Zero indicates that no records can be accessed through the Error Record System registers.

Each implemented record is owned by a node. A node might own multiple records.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERRIDR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERRIDR;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERRIDR;
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRSELR, Error Record Select Register

The ERRSELR characteristics are:

Purpose

Selects an error record to be accessed through the Error Record System registers.

Configuration

AArch32 System register ERRSELR bits [31:0] are architecturally mapped to AArch64 System register ERRSELR_EL1[31:0].

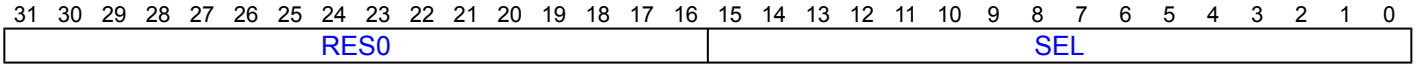
This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERRSELR are UNDEFINED.

If ERRIDR indicates that zero error records are implemented, then it is IMPLEMENTATION DEFINED whether ERRSELR is UNDEFINED or RES0.

Attributes

ERRSELR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

SEL, bits [15:0]

Selects the error record accessed through the ERX registers.

For example, if ERRSELR.SEL is `0x0004`, then direct reads and writes of ERXSTATUS access ERR4STATUS.

If ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then all of the following apply:

- The value read back from ERRSELR.SEL is UNKNOWN.
- One of the following occurs:
  - An UNKNOWN error record is selected.
  - The ERX* registers are RAZ/WI.
  - ERX* register reads and writes are NOPs.
  - ERX* register reads and writes are UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ERRSELR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERRSELR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERRSELR;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERRSELR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERRSELR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERRSELR = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERRSELR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXADDR, Selected Error Record Address Register

The ERXADDR characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>ADDR for the error record <n> selected by ERRSELR.SEL.

Configuration

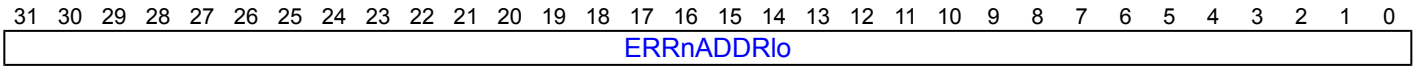
AArch32 System register ERXADDR bits [31:0] are architecturally mapped to AArch64 System register ERXADDR_EL1[31:0].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXADDR are UNDEFINED.

Attributes

ERXADDR is a 32-bit register.

Field descriptions



ERRnADDRlo, bits [31:0]

ERXADDR accesses bits [31:0] of ERR<n>ADDR, where <n> is the value in ERRSELR.SEL.

Accessing ERXADDR

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXADDR is RAZ/WI.
- Direct reads and writes of ERXADDR are NOPs.
- Direct reads and writes of ERXADDR are UNDEFINED.

ERR<n>ADDR describes additional constraints that also apply when ERR<n>ADDR is accessed through ERXADDR.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXADDR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXADDR;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXADDR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXADDR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXADDR = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXADDR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXADDR2, Selected Error Record Address Register 2

The ERXADDR2 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>ADDR for the error record <n> selected by ERRSELR.SEL.

Configuration

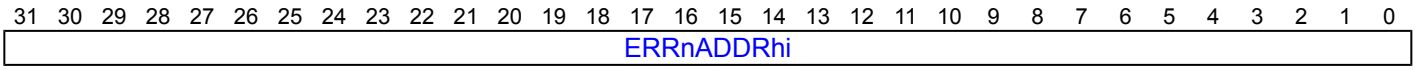
AArch32 System register ERXADDR2 bits [31:0] are architecturally mapped to AArch64 System register ERXADDR_EL1[63:32].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXADDR2 are UNDEFINED.

Attributes

ERXADDR2 is a 32-bit register.

Field descriptions



ERRnADDRhi, bits [31:0]

ERXADDR2 accesses bits [63:32] of ERR<n>ADDR, where <n> is the value in ERRSELR.SEL.

Accessing ERXADDR2

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXADDR2 is RAZ/WI.
- Direct reads and writes of ERXADDR2 are NOPs.
- Direct reads and writes of ERXADDR2 are UNDEFINED.

ERR<n>ADDR describes additional constraints that also apply when ERR<n>ADDR is accessed through ERXADDR2.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXADDR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXADDR2;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXADDR2;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXCTLR, Selected Error Record Control Register

The ERXCTLR characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>CTLR for the error record <n> selected by ERRSELR.SEL.

Configuration

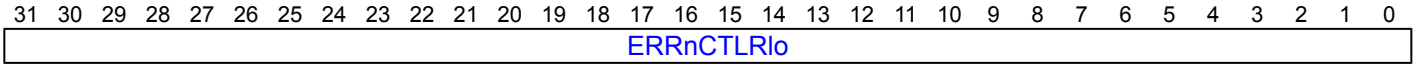
AArch32 System register ERXCTLR bits [31:0] are architecturally mapped to AArch64 System register ERXCTLR_EL1[31:0].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXCTLR are UNDEFINED.

Attributes

ERXCTLR is a 32-bit register.

Field descriptions



ERRnCTLRIo, bits [31:0]

ERXCTLR accesses bits [31:0] of ERR<n>CTLR, where <n> is the value in ERRSELR.SEL.

Accessing ERXCTLR

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXCTLR is RAZ/WI.
- Direct reads and writes of ERXCTLR are NOPs.
- Direct reads and writes of ERXCTLR are UNDEFINED.

If ERRSELR.SEL is not the index of the first error record owned by a node, then ERR<n>CTLR[31:0] is not present, meaning reads and writes of ERXCTLR are RES0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXCTLR;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXCTLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXCTLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXCTLR = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXCTLR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXCTLR2, Selected Error Record Control Register 2

The ERXCTLR2 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>CTLR for the error record <n> selected by ERRSELR.SEL.

Configuration

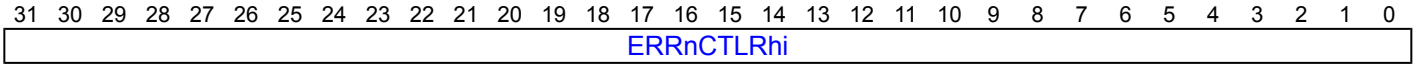
AArch32 System register ERXCTLR2 bits [31:0] are architecturally mapped to AArch64 System register ERXCTLR_EL1[63:32].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXCTLR2 are UNDEFINED.

Attributes

ERXCTLR2 is a 32-bit register.

Field descriptions



ERRnCTLRhi, bits [31:0]

ERXCTLR2 accesses bits [63:32] of ERR<n>CTLR, where <n> is the value in ERRSELR.SEL.

Accessing ERXCTLR2

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXCTLR2 is RAZ/WI.
- Direct reads and writes of ERXCTLR2 are NOPs.
- Direct reads and writes of ERXCTLR2 are UNDEFINED.

If ERRSELR.SEL is not the index of the first error record owned by a node, then ERR<n>CTLR[63:32] is not present, meaning reads and writes of ERXCTLR2 are RES0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0100	0b101

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXCTLR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXCTLR2;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXCTLR2;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0100	0b101

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXCTLR2 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXCTLR2 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXCTLR2 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXFR, Selected Error Record Feature Register

The ERXFR characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>FR for the error record <n> selected by ERRSELR.SEL.

Configuration

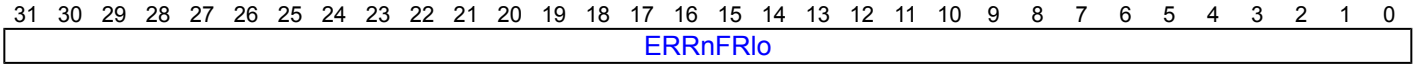
AArch32 System register ERXFR bits [31:0] are architecturally mapped to AArch64 System register ERXFR_EL1[31:0].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXFR are UNDEFINED.

Attributes

ERXFR is a 32-bit register.

Field descriptions



ERRnFRIo, bits [31:0]

ERXFR accesses bits [31:0] of ERR<n>FR, where <n> is the value in ERRSELR.SEL.

Accessing ERXFR

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXFR is RAZ.
- Direct reads of ERXFR are NOPs.
- Direct reads of ERXFR are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXFR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXFR;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXFR;
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXFR2, Selected Error Record Feature Register 2

The ERXFR2 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>FR for the error record <n> selected by ERRSELR.SEL.

Configuration

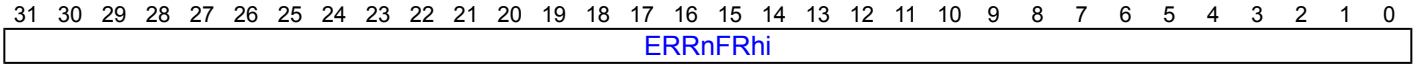
AArch32 System register ERXFR2 bits [31:0] are architecturally mapped to AArch64 System register ERXFR_EL1[63:32].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXFR2 are UNDEFINED.

Attributes

ERXFR2 is a 32-bit register.

Field descriptions



ERRnFRhi, bits [31:0]

ERXFR2 accesses bits [63:32] of ERR<n>FR, where <n> is the value in ERRSELR.SEL.

Accessing ERXFR2

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXFR2 is RAZ.
- Direct reads of ERXFR2 are NOPs.
- Direct reads of ERXFR2 are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0100	0b100

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXFR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXFR2;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXFR2;
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC0, Selected Error Record Miscellaneous Register 0

The ERXMISC0 characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>MISC0 for the error record <n> selected by ERRSELR.SEL.

Configuration

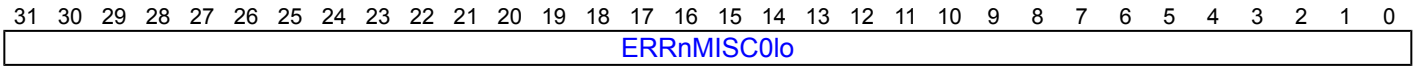
AArch32 System register ERXMISC0 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC0_EL1[31:0].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXMISC0 are UNDEFINED.

Attributes

ERXMISC0 is a 32-bit register.

Field descriptions



ERRnMISC0lo, bits [31:0]

ERXMISC0 accesses bits [31:0] of ERR<n>MISC0, where <n> is the value in ERRSELR.SEL.

Accessing ERXMISC0

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC0 is RAZ/WI.
- Direct reads and writes of ERXMISC0 are NOPs.
- Direct reads and writes of ERXMISC0 are UNDEFINED.

ERR<n>MISC0 describes additional constraints that also apply when ERR<n>MISC0 is accessed through ERXMISC0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0101	0b000

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC0;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC0;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0101	0b000

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC0 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC0 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC0 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC1, Selected Error Record Miscellaneous Register 1

The ERXMISC1 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>MISC0 for the error record <n> selected by ERRSELR.SEL.

Configuration

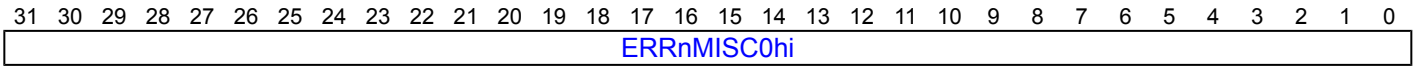
AArch32 System register ERXMISC1 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC0_EL1[63:32].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXMISC1 are UNDEFINED.

Attributes

ERXMISC1 is a 32-bit register.

Field descriptions



ERRnMISC0hi, bits [31:0]

ERXMISC1 accesses bits [63:32] of ERR<n>MISC0, where <n> is the value in ERRSELR.SEL.

Accessing ERXMISC1

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC1 is RAZ/WI.
- Direct reads and writes of ERXMISC1 are NOPs.
- Direct reads and writes of ERXMISC1 are UNDEFINED.

ERR<n>MISC0 describes additional constraints that also apply when ERR<n>MISC0 is accessed through ERXMISC1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC1;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC1;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC1 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC1 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC2, Selected Error Record Miscellaneous Register 2

The ERXMISC2 characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>MISC1 for the error record <n> selected by ERRSELR.SEL.

Configuration

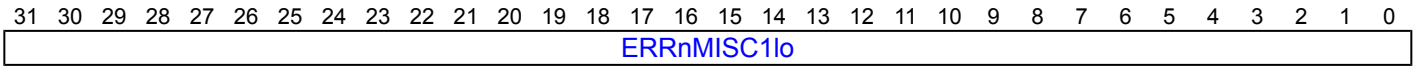
AArch32 System register ERXMISC2 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC1_EL1[31:0].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXMISC2 are UNDEFINED.

Attributes

ERXMISC2 is a 32-bit register.

Field descriptions



ERRnMISC1lo, bits [31:0]

ERXMISC2 accesses bits [31:0] of ERR<n>MISC1, where <n> is the value in ERRSELR.SEL.

Accessing ERXMISC2

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC2 is RAZ/WI.
- Direct reads and writes of ERXMISC2 are NOPs.
- Direct reads and writes of ERXMISC2 are UNDEFINED.

ERR<n>MISC1 describes additional constraints that also apply when ERR<n>MISC1 is accessed through ERXMISC2.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0101	0b100

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC2;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC2;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0101	0b100

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC2 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC2 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC2 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC3, Selected Error Record Miscellaneous Register 3

The ERXMISC3 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>MISC1 for the error record <n> selected by ERRSELR.SEL.

Configuration

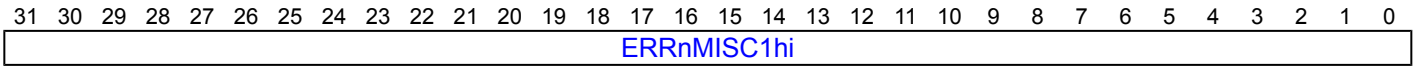
AArch32 System register ERXMISC3 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC1_EL1[63:32].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXMISC3 are UNDEFINED.

Attributes

ERXMISC3 is a 32-bit register.

Field descriptions



ERRnMISC1hi, bits [31:0]

ERXMISC3 accesses bits [63:32] of ERR<n>MISC1, where <n> is the value in ERRSELR.SEL.

Accessing ERXMISC3

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC3 is RAZ/WI.
- Direct reads and writes of ERXMISC3 are NOPs.
- Direct reads and writes of ERXMISC3 are UNDEFINED.

ERR<n>MISC1 describes additional constraints that also apply when ERR<n>MISC1 is accessed through ERXMISC3.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC3;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC3;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC3;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0101	0b101

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC3 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC3 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC3 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC4, Selected Error Record Miscellaneous Register 4

The ERXMISC4 characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>MISC2 for the error record <n> selected by ERRSELR.SEL.

Configuration

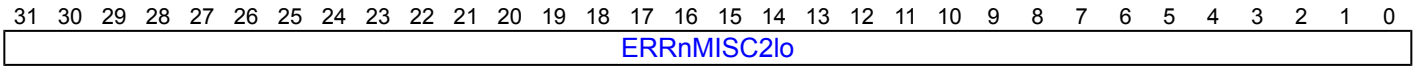
AArch32 System register ERXMISC4 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC2_EL1[31:0].

This register is present only when FEAT_RASv1p1 is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXMISC4 are UNDEFINED.

Attributes

ERXMISC4 is a 32-bit register.

Field descriptions



ERRnMISC2lo, bits [31:0]

ERXMISC4 accesses bits [31:0] of ERR<n>MISC2, where <n> is the value in ERRSELR.SEL.

Accessing ERXMISC4

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC4 is RAZ/WI.
- Direct reads and writes of ERXMISC4 are NOPs.
- Direct reads and writes of ERXMISC4 are UNDEFINED.

ERR<n>MISC2 describes additional constraints that also apply when ERR<n>MISC2 is accessed through ERXMISC4.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0101	0b010

```
if !(IsFeatureImplemented(FEAT_RASv1p1) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC4;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC4;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC4;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0101	0b010

```
if !(IsFeatureImplemented(FEAT_RASv1p1) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC4 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC4 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC4 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC5, Selected Error Record Miscellaneous Register 5

The ERXMISC5 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>MISC2 for the error record <n> selected by ERRSELR.SEL.

Configuration

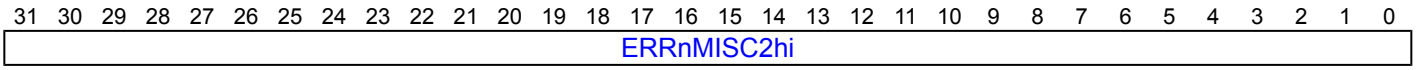
AArch32 System register ERXMISC5 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC2_EL1[63:32].

This register is present only when FEAT_RASv1p1 is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXMISC5 are UNDEFINED.

Attributes

ERXMISC5 is a 32-bit register.

Field descriptions



ERRnMISC2hi, bits [31:0]

ERXMISC5 accesses bits [63:32] of ERR<n>MISC2, where <n> is the value in ERRSELR.SEL.

Accessing ERXMISC5

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC5 is RAZ/WI.
- Direct reads and writes of ERXMISC5 are NOPs.
- Direct reads and writes of ERXMISC5 are UNDEFINED.

ERR<n>MISC2 describes additional constraints that also apply when ERR<n>MISC2 is accessed through ERXMISC5.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0101	0b011

```
if !(IsFeatureImplemented(FEAT_RASv1p1) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC5;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC5;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC5;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0101	0b011

```
if !(IsFeatureImplemented(FEAT_RASv1p1) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC5 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC5 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC5 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC6, Selected Error Record Miscellaneous Register 6

The ERXMISC6 characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>MISC3 for the error record <n> selected by ERRSELR.SEL.

Configuration

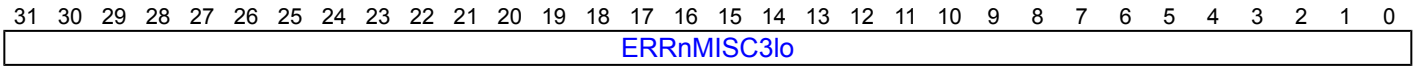
AArch32 System register ERXMISC6 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC3_EL1[31:0].

This register is present only when FEAT_RASv1p1 is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXMISC6 are UNDEFINED.

Attributes

ERXMISC6 is a 32-bit register.

Field descriptions



ERRnMISC3lo, bits [31:0]

ERXMISC6 accesses bits [31:0] of ERR<n>MISC3, where <n> is the value in ERRSELR.SEL.

Accessing ERXMISC6

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC6 is RAZ/WI.
- Direct reads and writes of ERXMISC6 are NOPs.
- Direct reads and writes of ERXMISC6 are UNDEFINED.

ERR<n>MISC3 describes additional constraints that also apply when ERR<n>MISC3 is accessed through ERXMISC6.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0101	0b110

```
if !(IsFeatureImplemented(FEAT_RASv1p1) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC6;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC6;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC6;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0101	0b110

```
if !(IsFeatureImplemented(FEAT_RASv1p1) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC6 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC6 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC6 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC7, Selected Error Record Miscellaneous Register 7

The ERXMISC7 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>MISC3 for the error record <n> selected by ERRSELR.SEL.

Configuration

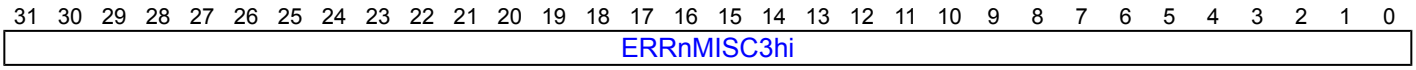
AArch32 System register ERXMISC7 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC3_EL1[63:32].

This register is present only when FEAT_RASv1p1 is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXMISC7 are UNDEFINED.

Attributes

ERXMISC7 is a 32-bit register.

Field descriptions



ERRnMISC3hi, bits [31:0]

ERXMISC7 accesses bits [63:32] of ERR<n>MISC3, where <n> is the value in ERRSELR.SEL.

Accessing ERXMISC7

If ERRIDR.NUM is `0x0000` or ERRSELR.SEL is greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC7 is RAZ/WI.
- Direct reads and writes of ERXMISC7 are NOPs.
- Direct reads and writes of ERXMISC7 are UNDEFINED.

ERR<n>MISC3 describes additional constraints that also apply when ERR<n>MISC3 is accessed through ERXMISC7.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0101	0b111

```
if !(IsFeatureImplemented(FEAT_RASv1p1) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC7;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC7;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXMISC7;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0101	0b111

```
if !(IsFeatureImplemented(FEAT_RASv1p1) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC7 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXMISC7 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC7 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXSTATUS, Selected Error Record Primary Status Register

The ERXSTATUS characteristics are:

Purpose

Accesses bits [31:0] of ERR<n>STATUS for the error record selected by ERRSELR.SEL.

Configuration

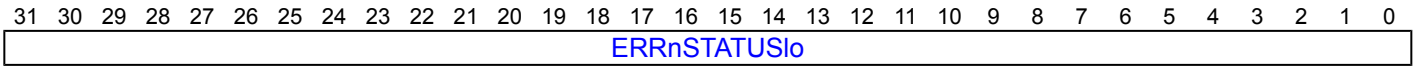
AArch32 System register ERXSTATUS bits [31:0] are architecturally mapped to AArch64 System register ERXSTATUS_EL1[31:0].

This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ERXSTATUS are UNDEFINED.

Attributes

ERXSTATUS is a 32-bit register.

Field descriptions



ERRnSTATUSIo, bits [31:0]

ERXSTATUS accesses bits [31:0] of ERR<n>STATUS, where n is the value in ERRSELR.SEL.

Accessing ERXSTATUS

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXSTATUS is RAZ/WI.
- Direct reads and writes of ERXSTATUS are NOPs.
- Direct reads and writes of ERXSTATUS are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXSTATUS;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXSTATUS;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = ERXSTATUS;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0101	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TERR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXSTATUS = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.TERR ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXSTATUS = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXSTATUS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FCSEIDR, FCSE Process ID register

The FCSEIDR characteristics are:

Purpose

Identifies whether the Fast Context Switch Extension (FCSE) is implemented.

From Armv8.0, the FCSE is not implemented, so this register is RAZ/WI. Software can access this register to determine that the implementation does not include the FCSE.

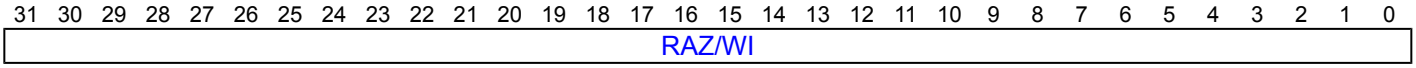
Configuration

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to FCSEIDR are UNDEFINED.

Attributes

FCSEIDR is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RAZ/WI.

Accessing FCSEIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = FCSEIDR;
elsif PSTATE.EL == EL2 then
    R[t] = FCSEIDR;
elsif PSTATE.EL == EL3 then
    R[t] = FCSEIDR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        FCSEIDR = R[t];
elsif PSTATE.EL == EL2 then
    FCSEIDR = R[t];
elsif PSTATE.EL == EL3 then
    FCSEIDR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FPEXC, Floating-Point Exception Control register

The FPEXC characteristics are:

Purpose

Provides a global enable for the implemented Advanced SIMD and floating-point functionality, and reports floating-point status information.

Configuration

AArch32 System register FPEXC bits [31:0] are architecturally mapped to AArch64 System register FPEXC32_EL2[31:0].

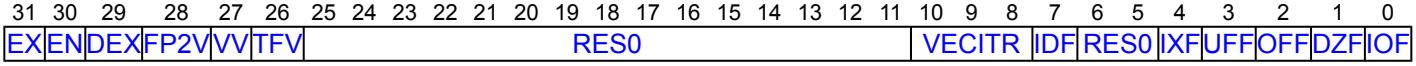
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to FPEXC are UNDEFINED.

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

Attributes

FPEXC is a 32-bit register.

Field descriptions



EX, bit [31]

Exception bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RAZ/WI**.

EN, bit [30]

Enables access to the Advanced SIMD and floating-point functionality from all Exception levels, except that setting this field to 0 does not disable the following:

- VMSR accesses to the FPEXC or FPSID.
- VMRS accesses from the FPEXC, FPSID, MVFR0, MVFR1, or MVFR2.

EN	Meaning
0b0	Accesses to the FPSCR, and any of the SIMD and floating-point registers Q0-Q15, including their views as D0-D31 registers or S0-S31 registers, are UNDEFINED at all Exception levels.
0b1	This control permits access to the Advanced SIMD and floating-point functionality at all Exception levels.

Execution of Advanced SIMD and floating-point instructions in AArch32 state can be disabled or trapped by the following controls:

- CPACR.cp10, or, if executing at EL0, CPACR_EL1.FPEN.
- FPEXC.EN.
- If executing in Non-secure state:
  - HCPTR.TCP10, or if EL2 is using AArch64, CPTR_EL2.TFP.
  - NSACR.cp10, or if EL3 is using AArch64, CPTR_EL3.TFP.
- For Advanced SIMD instructions only:
  - CPACR.ASEDIS.
  - If executing in Non-secure state, HCPTR.TASE and NSACR.NSASEDIS.

See the descriptions of the controls for more information.

Note

When executing at EL0 using AArch32:

- If EL1 is using AArch64, then the Effective value of FPEXC.EN is 1. This includes when EL2 is using AArch64 and is enabled in the current Security state, HCR_EL2.TGE is 1, and the Effective value of HCR_EL2.RW is 1.
- If EL2 is using AArch64 and is enabled in the current Security state, HCR_EL2.TGE is 1, and the Effective value of HCR_EL2.RW is 0, then it is IMPLEMENTATION DEFINED whether the Effective value of FPEXC.EN is 1 or the value written to FPEXC.EN. However, Arm deprecates using the value of FPEXC.EN to determine behavior.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

DEX, bit [29]

Defined synchronous exception on floating-point execution.

This field identifies whether a synchronous exception generated by the attempted execution of an instruction was generated by an unallocated encoding. The instruction must be in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr() returning TRUE. This field also indicates whether the FPEXC.TFV field is valid.

The meaning of this bit is:

DEX	Meaning
`0b0`	The exception was generated by the attempted execution of an unallocated instruction in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr(). If FPEXC.TFV is RW then it is invalid and UNKNOWN. If FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} are RW then they are invalid and UNKNOWN.
`0b1`	The exception was generated during the execution of an allocated encoding. FPEXC.TFV is valid and indicates the cause of the exception.

On an exception that sets this bit to 1 the exception-handling routine must clear this bit to 0.

On an implementation that both does not support trapping of floating-point exceptions and implements the FPSCR.{Stride, Len} fields as RAZ, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FP2V, bit [28]

FPINST2 instruction Valid bit. From Armv8.0, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RES0**.

VV, bit [27]

VECITR Valid bit. From Armv8.0, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RES0**.

TFV, bit [26]

Trapped Fault Valid bit. Valid only when the value of FPEXC.DEX is 1. When valid, it indicates the cause of the exception and therefore whether FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} are valid.

TFV	Meaning
0b0	The exception was caused by the execution of a floating-point VABS, VADD, VDIV, VFMA, VFMS, VFNMA, VFNMS, VMLA, VMLS, VMOV, VMUL, VNEG, VNMLA, VNMLS, VNMUL, VSQRT, or VSUB instruction when one or both of FPSCR.{Stride, Len} was nonzero. If FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} are RW then they are invalid and UNKNOWN.
0b1	FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} indicate the presence of trapped floating-point exceptions that had occurred at the time of the exception. Bits are set for all trapped exceptions that had occurred at the time of the exception.

This bit returns a status value and ignores writes.

When the value of FPEXC.DEX is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When an implementation does not implement trapping of floating-point exceptions, access to this field is **RAZ/WI**.
- When an implementation implements FPSCR.LEN,STRIDE as RAZ, access to this field is **RAO/WI**.

Bits [25:11]

Reserved, RES0.

VECITR, bits [10:8]

Vector Iteration count. From Armv8.0, this field is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RES1**.

IDF, bit [7]

Input Denormal trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Input Denormal exception occurred while FPSCR.IDE was 1:

IDF	Meaning
0b0	Input Denormal exception has not occurred.
0b1	Input Denormal exception has occurred.

Input Denormal exceptions can occur only when FPSCR.FZ is 1.

Note

A half-precision floating-point value that is flushed to zero because the value of FPSCR.FZ16 is 1 does not generate an Input Denormal exception.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Input Denormal floating-point exceptions, access to this field is **RAZ/WI**.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Inexact exception occurred while FPSCR.IXE was 1:

IXF	Meaning
0b0	Inexact exception has not occurred.
0b1	Inexact exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Inexact floating-point exceptions, access to this field is **RAZ/WI**.

UFF, bit [3]

Underflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Underflow exception occurred while FPSCR.UFE was 1:

UFF	Meaning
0b0	Underflow exception has not occurred.
0b1	Underflow exception has occurred.

Underflow trapped exceptions can occur:

- On half-precision data-processing instructions only when FPSCR.FZ16 is 0.
- Otherwise only when FPSCR.FZ is 0.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Underflow floating-point exceptions, access to this field is **RAZ/WI**.

OFF, bit [2]

Overflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Overflow exception occurred while FPSCR.OFE was 1:

OFF	Meaning
0b0	Overflow exception has not occurred.
0b1	Overflow exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Overflow floating-point exceptions, access to this field is **RAZ/WI**.

DZF, bit [1]

Divide by Zero trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether a Divide by Zero exception occurred while FPSCR.DZE was 1:

DZF	Meaning
0b0	Divide by Zero exception has not occurred.
0b1	Divide by Zero exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Divide by Zero floating-point exceptions, access to this field is **RAZ/WI**.

IOF, bit [0]

Invalid Operation trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Invalid Operation exception occurred while FPSCR.IOE was 1:

IOF	Meaning
0b0	Invalid Operation exception has not occurred.
0b1	Invalid Operation exception has occurred.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Invalid Operation floating-point exceptions, access to this field is **RAZ/WI**.

Accessing FPEXC

Accesses to this register use the following encodings in the System register encoding space:

```
VMRS{<c>}{<q>} <Rt>, <spec_reg>
```

reg
0b1000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = FPEXC;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS ==
'1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = FPEXC;
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        R[t] = FPEXC;
```

```
VMSR{<c>}{<q>} <spec_reg>, <Rt>
```

reg
0b1000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        FPEXC = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS ==
'1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        FPEXC = R[t];
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        FPEXC = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FPSCR, Floating-Point Status and Control Register

The FPSCR characteristics are:

Purpose

Provides floating-point system status information and control.

Configuration

AArch32 System register FPSCR bits [31:27] are architecturally mapped to AArch64 System register FPSR[31:27].

AArch32 System register FPSCR bit [7] is architecturally mapped to AArch64 System register FPSR[7].

AArch32 System register FPSCR bits [4:0] are architecturally mapped to AArch64 System register FPSR[4:0].

AArch32 System register FPSCR bits [26:15] are architecturally mapped to AArch64 System register FPCR[26:15].

AArch32 System register FPSCR bits [12:8] are architecturally mapped to AArch64 System register FPCR[12:8].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to FPSCR are UNDEFINED.

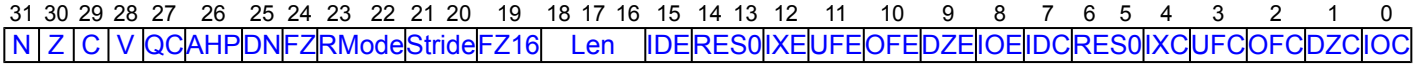
It is IMPLEMENTATION DEFINED whether the Len and Stride fields can be programmed to nonzero values, which will cause some AArch32 floating-point instruction encodings to be UNDEFINED, or whether these fields are RAZ.

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

Attributes

FPSCR is a 32-bit register.

Field descriptions



N, bit [31]

Negative condition flag. This is updated by floating-point comparison operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero condition flag. This is updated by floating-point comparison operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry condition flag. This is updated by floating-point comparison operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow condition flag. This is updated by floating-point comparison operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

QC, bit [27]

Cumulative saturation bit, Advanced SIMD only. This bit is set to 1 to indicate that an Advanced SIMD integer operation has saturated since 0 was last written to this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AHP, bit [26]

Alternative half-precision control bit:

AHP	Meaning
0b0	IEEE half-precision format selected.
0b1	Alternative half-precision format selected.

This bit is used only for conversions between half-precision floating-point and other floating-point formats.

The data-processing instructions added as part of the FEAT_FP16 extension always use the IEEE half-precision format, and ignore the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DN, bit [25]

Default NaN mode control bit:

DN	Meaning
0b0	NaN operands propagate through to the output of a floating-point operation.
0b1	Any operation involving one or more NaNs returns the Default NaN.

The value of this bit controls only scalar floating-point arithmetic. Advanced SIMD arithmetic always uses the Default NaN setting, regardless of the value of the DN bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FZ, bit [24]

Flush-to-zero mode control bit:

FZ	Meaning
0b0	Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard.
0b1	Flush-to-zero mode enabled.

The value of this bit controls only scalar floating-point arithmetic. Advanced SIMD arithmetic always uses the Flush-to-zero setting, regardless of the value of the FZ bit.

This bit has no effect on half-precision calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RMode, bits [23:22]

Rounding Mode control field. The encoding of this field is:

RMode	Meaning
0b00	Round to Nearest (RN) mode.
0b01	Round towards Plus Infinity (RP) mode.
0b10	Round towards Minus Infinity (RM) mode.
0b11	Round towards Zero (RZ) mode.

The specified rounding mode is used by almost all scalar floating-point instructions. Advanced SIMD arithmetic always uses the Round to Nearest setting, regardless of the value of the RMode bits.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Stride, bits [21:20]

If this field is RW and is set to a value other than zero, some floating-point instruction encodings are UNDEFINED. The instruction pseudocode identifies these instructions.

Arm strongly recommends that software never sets this field to a value other than zero.

The value of this field is ignored when processing Advanced SIMD instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation implements FPSCR.LEN,STRIDE as RAZ, access to this field is **RAZ/WI**.

FZ16, bit [19]

When FEAT_FP16 is implemented:

Flush-to-zero mode control bit on half-precision data-processing instructions:

FZ16	Meaning
0b0	Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard.
0b1	Flush-to-zero mode enabled.

The value of this bit applies to both scalar and Advanced SIMD floating-point half-precision calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Len, bits [18:16]

If this field is RW and is set to a value other than zero, some floating-point instruction encodings are UNDEFINED. The instruction pseudocode identifies these instructions.

Arm strongly recommends that software never sets this field to a value other than zero.

The value of this field is ignored when processing Advanced SIMD instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation implements FPSCR.LEN,STRIDE as RAZ, access to this field is **RAZ/WI**.

IDE, bit [15]

Input Denormal floating-point exception trap enable.

IDE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the IDC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the IDC bit.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Input Denormal floating-point exceptions, access to this field is **RAZ/WI**.

Bits [14:13]

Reserved, RES0.

IXE, bit [12]

Inexact floating-point exception trap enable.

IXE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the IXC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the IXC bit.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Inexact floating-point exceptions, access to this field is **RAZ/WI**.

UFE, bit [11]

Underflow floating-point exception trap enable.

UFE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the UFC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs and Flush-to-zero is not enabled, the PE does not update the UFC bit.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Underflow floating-point exceptions, access to this field is **RAZ/WI**.

OFE, bit [10]

Overflow floating-point exception trap enable.

OFE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the OFC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the OFC bit.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Overflow floating-point exceptions, access to this field is **RAZ/WI**.

DZE, bit [9]

Divide by Zero floating-point exception trap enable.

DZE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the DZC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the DZC bit.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Divide by Zero floating-point exceptions, access to this field is **RAZ/WI**.

IOE, bit [8]

Invalid Operation floating-point exception trap enable.

IOE	Meaning
0b0	Untrapped exception handling selected. If the floating-point exception occurs, the IOC bit is set to 1.
0b1	Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the IOC bit.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement trapping of Invalid Operation floating-point exceptions, access to this field is **RAZ/WI**.

IDC, bit [7]

Input Denormal cumulative floating-point exception bit. This bit is set to 1 to indicate that the Input Denormal floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the IDE bit.

Advanced SIMD instructions set this bit if the Input Denormal floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the IDE bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXC, bit [4]

Inexact cumulative floating-point exception bit. This bit is set to 1 to indicate that the Inexact floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the IXE bit.

Advanced SIMD instructions set this bit if the Inexact floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the IXE bit.

The criteria for the Inexact floating-point exception to occur are different in Flush-to-zero mode. For more information, see 'Flush-to-zero'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFC, bit [3]

Underflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Underflow floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the UFE bit.

Advanced SIMD instructions set this bit if the Underflow floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, if FPSCR.UFE is 0 or if Flush-to-zero is enabled.

The criteria for the Underflow floating-point exception to occur are different in Flush-to-zero mode. For more information, see 'Flush-to-zero'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFC, bit [2]

Overflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Overflow floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the OFE bit.

Advanced SIMD instructions set this bit if the Overflow floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the OFE bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZC, bit [1]

Divide by Zero cumulative floating-point exception bit. This bit is set to 1 to indicate that the Divide by Zero floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the DZE bit.

Advanced SIMD instructions set this bit if the Divide by Zero floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the DZE bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOC, bit [0]

Invalid Operation cumulative floating-point exception bit. This bit is set to 1 to indicate that the Invalid Operation floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the IOE bit.

Advanced SIMD instructions set this bit if the Invalid Operation floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the IOE bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FPSCR

Accesses to this register use the following encodings in the System register encoding space:

```
VMRS{<c>}{<q>} <Rt>, <spec_reg>
```

reg
0b0001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x00);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x07);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| CPACR.cp10 IN {'0x'}) then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && ELIsInHost(EL0) &&
CPTR_EL2.FPEN != '11' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && ELIsInHost(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1)) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = FPSCR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = FPSCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS ==
'1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = FPSCR;
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        R[t] = FPSCR;
```

```
VMSR{<c>}{<q>} <spec_reg>, <Rt>
```

reg
0b0001

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && !ELIsInHost(EL0) &&
CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x00);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x07);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| CPACR.cp10 IN {'0x'}) then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && ELIsInHost(EL0) &&
CPTR_EL2.FPEN != '11' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && ELIsInHost(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1)) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        FPSCR = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        FPSCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS ==
'1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        FPSCR = R[t];
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        FPSCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FPSID, Floating-Point System ID register

The FPSID characteristics are:

Purpose

Provides top-level information about the floating-point implementation.

This register largely duplicates information held in the MIDR. Arm deprecates use of it.

Configuration

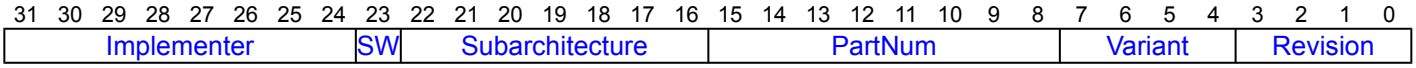
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to FPSID are UNDEFINED.

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

Attributes

FPSID is a 32-bit register.

Field descriptions



Implementer, bits [31:24]

Implementer codes are the same as those used for the MIDR.

For an implementation by Arm this field is `0x41`, the ASCII code for A.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SW, bit [23]

Software bit.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SW	Meaning
`0b0`	The implementation provides a hardware implementation of the floating-point instructions.
`0b1`	The implementation supports only software emulation of the floating-point instructions.

In Armv8-A, the only permitted value is `0b0`.

Access to this field is **RO**.

Subarchitecture, bits [22:16]

Subarchitecture version number.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Subarchitecture	Meaning
0b0000000	VFPv1 architecture with an IMPLEMENTATION DEFINED subarchitecture.
0b0000001	VFPv2 architecture with Common VFP subarchitecture v1.
0b0000010	VFPv3 architecture, or later, with Common VFP subarchitecture v2. The VFP architecture version is indicated by the MVFR0 and MVFR1 registers.
0b0000011	VFPv3 architecture, or later, with Null subarchitecture. The entire floating-point implementation is in hardware, and no software support code is required. The VFP architecture version is indicated by the MVFR0 and MVFR1 registers. This value can be used only by an implementation that does not support the trap enable bits in the FPSCR.
0b0000100	VFPv3 architecture, or later, with Common VFP subarchitecture v3, and support for trap enable bits in FPSCR. The VFP architecture version is indicated by the MVFR0 and MVFR1 registers.

For a subarchitecture designed by Arm the most significant bit of this field, register bit[22], is 0. Values with a most significant bit of 0 that are not listed here are reserved.

When the subarchitecture designer is not Arm, the most significant bit of this field, register bit[22], must be 1. Each implementer must maintain its own list of subarchitectures it has designed, starting at subarchitecture version number 0x40.

In Armv8-A, the permitted values are 0b0000011 and 0b0000100.

Access to this field is **RO**.

PartNum, bits [15:8]

Part Number for the floating-point implementation, assigned by the implementer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [7:4]

Variant number. Typically, this field distinguishes between different production variants of a single product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [3:0]

Revision number for the floating-point implementation.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing FPSID

Accesses to this register use the following encodings in the System register encoding space:

```
VMRS{<c>}{<q>} <Rt>, <spec_reg>
```

reg
0b0000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID0 ==
'1' then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = FPSID;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS ==
'1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = FPSID;
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        R[t] = FPSID;
```

```
VMSR{<c>}{<q>} <spec_reg>, <Rt>
```

reg
0b0000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID0 ==
'1' then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        return;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS ==
'1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        return;
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        return;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HACR, Hyp Auxiliary Configuration Register

The HACR characteristics are:

Purpose

Controls trapping to Hyp mode of IMPLEMENTATION DEFINED aspects of Non-secure EL1 or EL0 operation.

Configuration

AArch32 System register HACR bits [31:0] are architecturally mapped to AArch64 System register HACR_EL2[31:0].

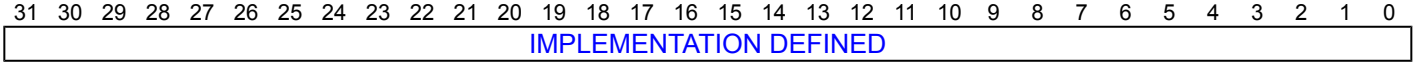
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HACR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HACR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HACR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HACR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HACR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HACR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HACR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HACTLR, Hyp Auxiliary Control Register

The HACTLR characteristics are:

Purpose

Controls IMPLEMENTATION DEFINED features of Hyp mode operation.

Configuration

AArch32 System register HACTLR bits [31:0] are architecturally mapped to AArch64 System register ACTLR_EL2[31:0].

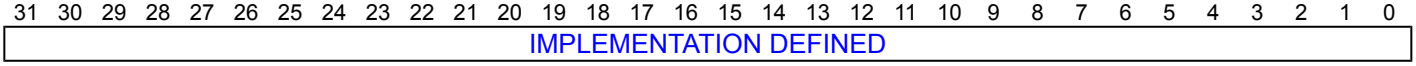
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HACTLR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HACTLR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HACTLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HACTLR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HACTLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HACTLR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HACTLR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HACTLR2, Hyp Auxiliary Control Register 2

The HACTLR2 characteristics are:

Purpose

Provides additional space to the HACTLR register to hold IMPLEMENTATION DEFINED trap functionality.

Configuration

AArch32 System register HACTLR2 bits [31:0] are architecturally mapped to AArch64 System register ACTLR_EL2[63:32].

This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HACTLR2 are UNDEFINED.

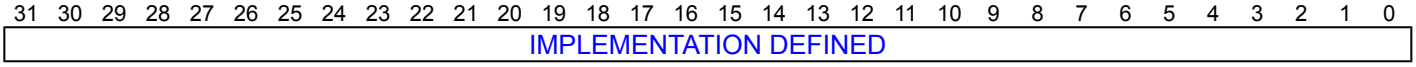
In Armv8.0 and Armv8.1, it is IMPLEMENTATION DEFINED whether this register is implemented, or whether it causes UNDEFINED exceptions when accessed. The implementation of this register can be detected by examining ID_MMFR4.AC2.

From Armv8.2 this register must be implemented.

Attributes

HACTLR2 is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HACTLR2

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HACTLR2;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HACTLR2;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HACTLR2 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HACTLR2 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HADFSR, Hyp Auxiliary Data Fault Status Register

The HADFSR characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED syndrome information for Data Abort exceptions taken to Hyp mode.

Configuration

AArch32 System register HADFSR bits [31:0] are architecturally mapped to AArch64 System register AFSR0_EL2[31:0].

This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HADFSR are UNDEFINED.

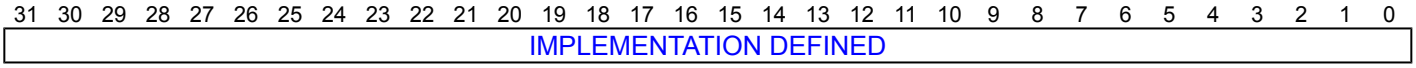
This is an optional register. An implementation that does not require this register can implement it as RES0.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HADFSR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HADFSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HADFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HADFSR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0101	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HADFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HADFSR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HAIFSR, Hyp Auxiliary Instruction Fault Status Register

The HAIFSR characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED syndrome information for Prefetch Abort exceptions taken to Hyp mode.

Configuration

AArch32 System register HAIFSR bits [31:0] are architecturally mapped to AArch64 System register AFSR1_EL2[31:0].

This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HAIFSR are UNDEFINED.

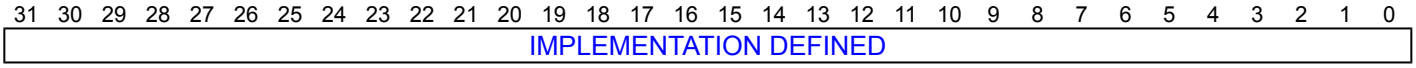
This is an optional register. An implementation that does not require this register can implement it as RES0.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HAIFSR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HAIFSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HAIFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HAIFSR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0101	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HAIFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HAIFSR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0

The HAMAIR0 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory attribute encodings defined by HMAIR0. These IMPLEMENTATION DEFINED attributes can only provide additional qualifiers for the memory attribute encodings, and cannot change the memory attributes defined in HMAIR0.

Configuration

AArch32 System register HAMAIR0 bits [31:0] are architecturally mapped to AArch64 System register AMAIR_EL2[31:0].

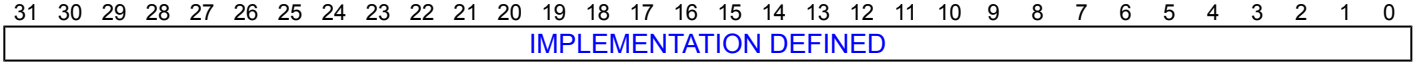
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HAMAIR0 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HAMAIR0 is a 32-bit register.

Field descriptions



If an implementation does not provide any IMPLEMENTATION DEFINED memory attributes, this register is RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HAMAIR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HAMAIR0;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HAMAIR0;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1010	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HAMAIR0 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HAMAIR0 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1

The HAMAIR1 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory attribute encodings defined by HMAIR1. These IMPLEMENTATION DEFINED attributes can only provide additional qualifiers for the memory attribute encodings, and cannot change the memory attributes defined in HMAIR1.

Configuration

AArch32 System register HAMAIR1 bits [31:0] are architecturally mapped to AArch64 System register AMAIR_EL2[63:32].

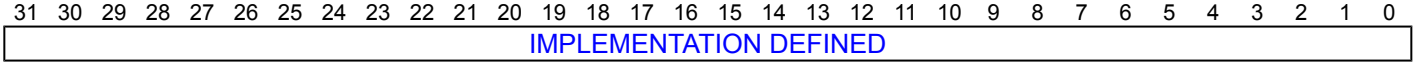
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HAMAIR1 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HAMAIR1 is a 32-bit register.

Field descriptions



If an implementation does not provide any IMPLEMENTATION DEFINED memory attributes, this register is RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HAMAIR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1010	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HAMAIR1;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HAMAIR1;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1010	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HAMAIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HAMAIR1 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HCPTR, Hyp Architectural Feature Trap Register

The HCPTR characteristics are:

Purpose

Controls:

- Trapping to Hyp mode of Non-secure access, at EL1 or EL0, to trace, and to Advanced SIMD and floating-point functionality.
- Hyp mode access to trace, and to Advanced SIMD and floating-point functionality.

Note

Accesses to this functionality:

- From Non-secure modes other than Hyp mode are also affected by settings in the CPACR and NSACR.
- From Hyp mode are also affected by settings in the NSACR.

Exceptions generated by the CPACR and NSACR controls are higher priority than those generated by the HCPTR controls.

Configuration

AArch32 System register HCPTR bits [31:0] are architecturally mapped to AArch64 System register CPTR_EL2[31:0].

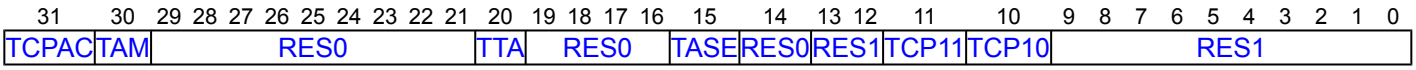
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HCPTR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HCPTR is a 32-bit register.

Field descriptions



TCPAC, bit [31]

Traps Non-secure EL1 MRC and MCR accesses to the CPACR to Hyp mode, reported using EC syndrome value `0x03`.

TCPAC	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Non-secure EL1 accesses to the CPACR are trapped to Hyp mode.

Note

The CPACR is not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Trap Activity Monitor access. Traps Non-secure EL1 and EL0 MRC, MCR, MRRC, and MCRR accesses to all Activity Monitor registers to EL2, reported using EC syndrome values `0x03` and `0x04`.

TAM	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Accesses from Non-secure EL1 and EL0 to Activity Monitor registers are trapped to Hyp mode.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

Traps Non-secure System register MRC, MCR, MRRC, and MCRR accesses to all implemented trace registers to Hyp mode, reported using EC syndrome values `0x05` and `0x0C`.

TTA	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Any Non-secure System register access to an implemented trace register is trapped to Hyp mode, unless the access is trapped to EL1 by a CPACR or NSACR control, or the access is from Non-secure EL0 and the definition of the register in the appropriate trace architecture specification indicates that the register is not accessible from EL0. A trapped instruction generates: • A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1. • An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode.

If the implementation does not include a trace unit, or does not include a System register interface to the trace unit registers, it is IMPLEMENTATION DEFINED whether this bit:

- Is RES0.
- Is RES1.
- Can be written from Hyp mode, and from Secure Monitor mode when SCR.NS is 1.

If EL3 is implemented and is using AArch32, and the value of NSACR.NSTRCDIS is 1, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

Note

- The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED, and a resulting Undefined Instruction exception is higher priority than a HCPTR.TTA Hyp Trap exception.
- The Arm architecture does not provide traps on trace register accesses through the optional memory-mapped debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [19:16]

Reserved, RES0.

TASE, bit [15]

Traps Non-secure execution of Advanced SIMD instructions to Hyp mode, reported using EC syndrome value 0x07, when the value of HCPTR.TCP10 is 0.

TASE	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	When the value of HCPTR.TCP10 is 0, any attempt to execute an Advanced SIMD instruction in Non-secure state is trapped to Hyp mode, unless it is trapped to EL1 by a CPACR or NSACR control. A trapped instruction generates: • A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1. • An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode.

When the value of HCPTR.TCP10 is 1, the value of this field is ignored.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1. Otherwise, it is IMPLEMENTATION DEFINED whether this field is implemented as a RW field. If it is not implemented as a RW field, then it is RAZ/WI.

If EL3 is implemented and is using AArch32, and the value of NSACR.NSASEDIS is 1, in Non-secure state this field behaves as RAO/WI, regardless of its actual value. This applies even if the field is implemented as RAZ/WI.

For the list of instructions affected by this field, see 'Controls of Advanced SIMD operation that do not apply to floating-point operation'.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [14]

Reserved, RES0.

Bits [13:12]

Reserved, RES1.

TCP11, bit [11]

When FEAT_FP is implemented and FEAT_AdvSIMD is implemented:

The value of this field is ignored. If this field is programmed with a different value to the TCP10 bit then this field is UNKNOWN on a direct read of the HCPTR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1.

If EL3 is implemented and is using AArch32, and the value of NSACR.cp10 is 0, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

The reset behavior of this field is:

- On a Warm reset:
    - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
    - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAO/WI** if all the following are true:
    - EL3 is implemented.
    - EL3 is using AArch32.
    - !IsCurrentSecurityState(SS_Secure).
    - NSACR.cp10 == 0.

Otherwise:

Reserved, RES1.

TCP10, bit [10]

When FEAT_FP is implemented and FEAT_AdvSIMD is implemented:

Trap Non-secure accesses to Advanced SIMD and floating-point functionality to Hyp mode, reported using EC syndrome value `0x07`:

TCP10	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Any attempted access to Advanced SIMD and floating-point functionality from Non-secure state is trapped to Hyp mode, unless it is trapped to EL1 by a CPACR or NSACR control. A trapped instruction generates: • A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1. • An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode.

The Advanced SIMD and floating-point features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the FPSCR, FPSID, MVFR0, MVFR1, MVFR2, or FPEXC System registers.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1.

If EL3 is implemented and is using AArch32, and the value of NSACR.cp10 is 0, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

The reset behavior of this field is:

- On a Warm reset:
    - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
    - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAO/WI** if all the following are true:
    - EL3 is implemented.
    - EL3 is using AArch32.
    - !IsCurrentSecurityState(SS_Secure).
    - NSACR.cp10 == 0.

Otherwise:

Reserved, RES1.

Bits [9:0]

Reserved, RES1.

Accessing HCPTR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = HCPTR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HCPTR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TCPAC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        HCPTR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HCPTR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HCR, Hyp Configuration Register

The HCR characteristics are:

Purpose

Provides configuration controls for virtualization, including defining whether various Non-secure operations are trapped to Hyp mode.

Configuration

AArch32 System register HCR bits [31:0] are architecturally mapped to AArch64 System register HCR_EL2[31:0].

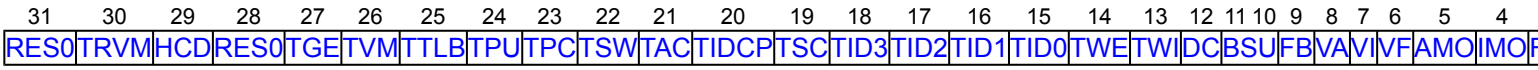
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HCR is a 32-bit register.

Field descriptions



Bit [31]

Reserved, RES0.

TRVM, bit [30]

Trap Reads of Virtual Memory controls. Traps Non-secure EL1 reads of the virtual memory control registers to EL2, when EL2 is enabled in the current Security state.

MRC reads of the following registers are trapped and reported using EC syndrome value `0x03` and MRRC reads are trapped and reported using EC syndrome value `0x04`:

SCTLR, TTBR0, TTBR1, TTBCR, TTBCR2, DACR, DFSR, IFSR, DFAR, IFAR, ADFSR, AIFSR, PRRR, NMRR, MAIR0, MAIR1, AMAIR0, AMAIR1, CONTEXTIDR.

TRVM	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Non-secure EL1 read accesses to the specified Virtual Memory controls are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

HCD, bit [29]

When EL3 is not implemented:

HVC instruction disable. Disables Non-secure EL1 and EL2 execution of HVC instructions, when EL2 is enabled in the current Security state, reported using EC syndrome value `0x00`.

HCD	Meaning
0b0	HVC instruction execution is enabled at EL2 and EL1.
0b1	HVC instructions are UNDEFINED at EL2 and Non-secure EL1. The Undefined Instruction exception is taken to the Exception level at which the HVC instruction is executed.

Note

HVC instructions are always UNDEFINED at EL0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [28]

Reserved, RES0.

TGE, bit [27]

Trap General Exceptions, from Non-secure EL0.

TGE	Meaning
0b0	This control has no effect on execution at EL0.
0b1	When EL2 is not enabled in the current Security state, this control has no effect on execution at EL0. When EL2 is enabled in the current Security state, then: • All exceptions that would be routed to EL1 are routed to EL2. • The SCTLR.M bit is treated as being 0 for all purposes other than returning the result of a direct read of SCTLR. • The HCR.{FMO, IMO, AMO} bits are treated as being 1 for all purposes other than returning the result of a direct read of HCR. • All virtual interrupts are disabled. • Any IMPLEMENTATION DEFINED mechanisms for signaling virtual interrupts are disabled. • An exception return to EL1 is treated as an illegal exception return. • Monitor mode execution of an MSR or CPS instruction that changes PSTATE.M to a Non-secure EL1 mode is an illegal change to PSTATE.M. For more information see 'Illegal changes to PSTATE.M'.

Also, when HCR.TGE is 1:

- If EL3 is using AArch32, an attempt to change from a Secure PL1 mode to a Non-secure EL1 mode by changing SCR.NS from 0 to 1 results in SCR.NS remaining as 0.
- The HDCR.{TDRA, TDOSA, TDA, TDE} bits are ignored and treated as being 1 other than for the purpose of a direct read of HDCR.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TVM, bit [26]

Trap Virtual Memory controls. Traps Non-secure EL1 writes to the virtual memory control registers to EL2, when EL2 is enabled in the current Security state.

MCR writes of the following registers are trapped and reported using EC syndrome value `0x03` and MCRR writes are trapped and reported using EC syndrome value `0x04`:

SCTLR, TTBR0, TTBR1, TTBCR, TTBCR2, DACR, DFSR, IFSR, DFAR, IFAR, ADFSR, AIFSR, PRRR, NMRR, MAIR0, MAIR1, AMAIR0, AMAIR1, CONTEXTIDR.

TVM	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Non-secure EL1 write accesses to the specified virtual memory control registers are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TTLB, bit [25]

Trap TLB maintenance instructions. Traps Non-secure EL1 execution of a TLBI instruction to EL2, when EL2 is enabled in the current Security state.

MCR and MRC accesses to the following system instructions are trapped and reported using EC syndrome value `0x03`:

TLBIALLIS, TLBIMVAIS, TLBIASIDIS, TLBIMVAAIS, TLBIMVALIS, TLBIMVAALIS, ITLBIALL, ITLBIMVA, ITLBIASID, DTLBIALL, DTLBIMVA, DTLBIASID, TLBIALL, TLBIMVA, TLBIASID, TLBIMVAA, TLBIMVAL, TLBIMVAAL

TTLB	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Non-secure EL1 accesses to the specified TLB maintenance instructions are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPU, bit [24]

Trap cache maintenance instructions that operate to the Point of Unification. Traps Non-secure EL1 execution of those cache maintenance instructions to EL2, when EL2 is enabled in the current Security state.

MRC and MCR accesses of the following system instructions are trapped and reported using EC syndrome value `0x03`:

- ICIMVAU, ICIALLU, ICIALLUIS, DCCMVAU.

Note

An Undefined Instruction exception generated at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.

TPU	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

TPC, bit [23]

Trap data or unified cache maintenance instructions that operate to the Point of Coherency. Traps Non-secure EL1 execution of those cache maintenance instructions to EL2, when EL2 is enabled in the current Security state.

MRC and MCR accesses of the following system instructions are trapped and reported using EC syndrome value 0x03:

- DCIMVAC, DCCIMVAC, DCCMVAC.

Note

An Undefined Instruction exception generated at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.

TPC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2.

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, invalidate, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TSW, bit [22]

Trap data or unified cache maintenance instructions that operate by Set/Way. Traps Non-secure EL1 execution of those cache maintenance instructions by set/way to EL2, when EL2 is enabled in the current Security state.

MRC and MCR accesses of the following system instructions are trapped and reported using EC syndrome value 0x03:

- DCISW, DCCSW, DCCISW.

Note

An Undefined Instruction exception generated at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.

TSW	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TAC, bit [21]

Trap Auxiliary Control Registers. Traps Non-secure EL1 accesses to the Auxiliary Control Registers to EL2, when EL2 is enabled in the current Security state, from both Execution states.

MRC and MCR accesses of the following registers are trapped and reported using EC syndrome value 0x03:

ACTLR and, if implemented, ACTLR2.

TAC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL1 accesses to the specified registers are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TIDCP, bit [20]

Trap IMPLEMENTATION DEFINED functionality. Traps Non-secure EL1 accesses to the encodings for IMPLEMENTATION DEFINED System Registers to EL2, when EL2 is enabled in the current Security state.

MRC and MCR accesses of the following encodings are trapped and reported using EC syndrome value 0x03:

- All coproc==p15, CRn==c9, Opcode1 = {0-7}, CRm == {c0-c2, c5-c8}, opcode2 == {0-7}.
- All coproc==p15, CRn==c10, Opcode1 =={0-7}, CRm == {c0, c1, c4, c8}, opcode2 == {0-7}.
- All coproc==p15, CRn==c11, Opcode1=={0-7}, CRm == {c0-c8, c15}, opcode2 == {0-7}.

TIDCP	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL1 accesses to the specified System register encodings for IMPLEMENTATION DEFINED functionality are trapped to EL2.

When HCR.TIDCP is set to 1, it is IMPLEMENTATION DEFINED whether any of this functionality accessed from Non-secure EL0 is trapped to EL2. Otherwise, it is UNDEFINED and the PE takes an Undefined Instruction exception to Non-secure Undefined mode.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TSC, bit [19]

Trap SMC instructions. Traps Non-secure EL1 execution of SMC instructions to Hyp mode.

TSC	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute an SMC instruction at Non-secure EL1 is trapped to Hyp mode, regardless of the value of SCR.SCD.

The Armv8-A architecture permits, but does not require, this trap to apply to conditional SMC instructions that fail their condition code check, in the same way as with traps on other conditional instructions.

Note

- This trap is implemented only if the implementation includes EL3.
- SMC instructions are always UNDEFINED at PL0.
- This bit traps execution of the SMC instruction, reported using EC syndrome value 0x13. It is not a routing control for the SMC exception. Hyp Trap exceptions and SMC exceptions have different preferred return addresses.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TID3, bit [18]

Trap ID group 3. Traps Non-secure EL1 reads of the following registers to EL2, when EL2 is enabled in the current Security state as follows:

- VMRS access to MVFR0, MVFR1, and MVFR2, reported using EC syndrome value `0x08`, unless access is also trapped by HCPTR which takes priority.
- MRC access to the following registers are reported using EC syndrome value `0x03`:
  - ID_PFR0, ID_PFR1, ID_PFR2, ID_DFR0, ID_AFR0, ID_MMFR0, ID_MMFR1, ID_MMFR2, ID_MMFR3, ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, ID_ISAR4, and ID_ISAR5.
  - If FEAT_FGT is implemented:
    - ID_MMFR4 and ID_MMFR5 are trapped to EL2.
    - ID_ISAR6 is trapped to EL2.
    - ID_DFR1 is trapped to EL2.
    - This field traps all MRC accesses to registers in the following range that are not already mentioned in this field description: coproc == p15, opc1 == 0, CRn == c0, CRm == {c2-c7}, opc2 == {0-7}.
  - If FEAT_FGT is not implemented:
    - ID_MMFR4 and ID_MMFR5 are trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to ID_MMFR4 or ID_MMFR5 are trapped.
    - ID_ISAR6 is trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to ID_ISAR6 are trapped to EL2.
    - ID_DFR1 is trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to ID_DFR1 are trapped to EL2.
    - Otherwise, it is IMPLEMENTATION DEFINED whether this bit traps MRC accesses to registers not already mentioned, with coproc == p15, opc1 == 0, CRn == c0, CRm == {c2-c7}, opc2 == {0-7}.

TID3	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	The specified Non-secure EL1 read accesses to ID group 3 registers are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TID2, bit [17]

Trap ID group 2. Traps the following register MRC and MCR accesses to EL2, reported using EC syndrome value `0x03`, when EL2 is enabled in the current Security state:

- Non-secure EL1 and EL0 reads of the CTR, CCSIDR, CCSIDR2, CLIDR, and CSSELR.
- Non-secure EL1 and EL0 writes to the CSSELR.

TID2	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	The specified Non-secure EL1 and EL0 accesses to ID group 2 registers are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TID1, bit [16]

Trap ID group 1. Traps Non-secure EL1 MRC reads of the following registers to EL2, reported using EC syndrome value `0x03`, when EL2 is enabled in the current Security state:

TCMTR, TLBTR, REVIDR, AIDR.

TID1	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	The specified Non-secure EL1 read accesses to ID group 1 registers are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TID0, bit [15]

Trap ID group 0. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

- Non-secure EL1 VMRS reads of FPSID reported using EC syndrome value `0x08`.
- Non-secure EL0 and EL1 MCR and MRC accesses of JIDR reported using EC syndrome value `0x05`.

Note

- It is IMPLEMENTATION DEFINED whether the JIDR is RAZ or UNDEFINED at EL0. If it is UNDEFINED at EL0 then the Undefined Instruction exception takes precedence over this trap.
- The FPSID is not accessible at EL0.
- Writes to the FPSID are ignored, and not trapped by this control.

TID0	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	The specified Non-secure EL1 read accesses to ID group 0 registers are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TWE, bit [14]

Traps Non-secure EL0 and EL1 execution of WFE instructions to EL2, reported using EC syndrome value `0x01`, when EL2 is enabled in the current Security state.

TWE	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Any attempt to execute a WFE instruction at Non-secure EL0 or EL1 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE.

The attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE can complete at any time, even without a Wakeup event, the traps on WFE are not guaranteed to be taken, even if the WFE is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TWI, bit [13]

Traps Non-secure EL0 and EL1 execution of WFI instructions to EL2, reported using EC syndrome value 0x01, when EL2 is enabled in the current Security state.

TWI	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFI instruction at Non-secure EL0 or EL1 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI.

The attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFI can complete at any time, even without a Wakeup event, the traps on WFI are not guaranteed to be taken, even if the WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DC, bit [12]

Default Cacheability.

DC	Meaning
0b0	This control has no effect on the Non-secure EL1&0 translation regime.
0b1	In Non-secure state: • The SCTLR.M field behaves as 0 for all purposes other than a direct read of the value of the field. • The HCR.VM field behaves as 1 for all purposes other than a direct read of the value of the field. • The memory type produced by the first stage of the EL1&0 translation regime is Normal Non-Shareable, Inner Write-Back Read-Allocate Write-Allocate, Outer Write-Back Read-Allocate Write-Allocate.

This field has no effect on the EL2 and EL3 translation regimes.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

BSU, bits [11:10]

Barrier Shareability upgrade. This field determines the minimum shareability domain that is applied to any barrier instruction executed from Non-secure EL1 or Non-secure EL0:

BSU	Meaning
0b00	No effect.
0b01	Inner Shareable.
0b10	Outer Shareable.
0b11	Full system.

This value is combined with the specified level of the barrier held in its instruction, using the same principles as combining the shareability attributes from two stages of address translation.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

FB, bit [9]

Force broadcast. Causes the following instructions to be broadcast within the Inner Shareable domain when executed from Non-secure EL1:

BPIALL, TLBIALL, TLBIMVA, TLBIASID, DTLBIALL, DTLBIMVA, DTLBIASID, ITLBIALL, ITLBIMVA, ITLBIASID, TLBIMVAA, ICIALLU, TLBIMVAL, TLBIMVAAL.

FB	Meaning
0b0	This field has no effect on the operation of the specified instructions.
0b1	When one of the specified instruction is executed at Non-secure EL1, the instruction is broadcast within the Inner Shareable shareability domain.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

VA, bit [8]

Virtual SError exception.

VA	Meaning
0b0	This mechanism is not making a virtual SError exception pending.
0b1	A virtual SError exception is pending because of this mechanism.

The virtual SError exception is enabled only when the value of HCR.{TGE, AMO} is {0, 1}.

The Guest OS cannot distinguish the virtual exception from the corresponding physical exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

VI, bit [7]

Virtual IRQ exception.

VI	Meaning
0b0	This mechanism is not making a virtual IRQ pending.
0b1	A virtual IRQ is pending because of this mechanism.

The virtual IRQ is enabled only when the value of HCR.{TGE, IMO} is {0, 1}.

The Guest OS cannot distinguish the virtual exception from the corresponding physical exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

VF, bit [6]

Virtual FIQ exception.

VF	Meaning
0b0	This mechanism is not making a virtual FIQ pending.
0b1	A virtual FIQ is pending because of this mechanism.

The virtual FIQ is enabled only when the value of HCR.{TGE, FMO} is {0, 1}.

The Guest OS cannot distinguish the virtual exception from the corresponding physical exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AMO, bit [5]

SError exception Mask Override. When this bit is set to 1, it overrides the effect of PSTATE.A, and enables virtual exception signaling by the VA bit.

If the value of HCR.TGE is 0, then virtual SError exceptions are enabled in Non-secure state.

If the value of HCR.TGE is 1, then in Non-secure state the HCR.AMO bit behaves as 1 for all purposes other than a direct read of the value of the bit.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

IMO, bit [4]

IRQ Mask Override. When this bit is set to 1, it overrides the effect of PSTATE.I, and enables virtual exception signaling by the VI bit.

If the value of HCR.TGE is 0, then Virtual IRQ interrupts are enabled in the Non-secure state.

If the value of HCR.TGE is 1, then in Non-secure state the HCR.IMO bit behaves as 1 for all purposes other than a direct read of the value of the bit.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

FMO, bit [3]

FIQ Mask Override. When this bit is set to 1, it overrides the effect of PSTATE.F, and enables virtual exception signaling by the VF bit.

If the value of HCR.TGE is 0, then Virtual FIQ interrupts are enabled in the Non-secure state.

If the value of HCR.TGE is 1, then in Non-secure state the HCR.FMO bit behaves as 1 for all purposes other than a direct read of the value of the bit.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

PTW, bit [2]

Protected Table Walk. In the Non-secure PL1&0 translation regime, a translation table access made as part of a stage 1 translation table walk is subject to a stage 2 translation. The combining of the memory type attributes from the two stages of translation means the access might be made to a type of Device memory. If this occurs then the value of this bit determines the behavior:

PTW	Meaning
0b0	The translation table walk occurs as if it is to Normal Non-cacheable memory. This means it can be made speculatively.
0b1	The memory access generates a stage 2 Permission fault.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SWIO, bit [1]

Set/Way Invalidation Override. Causes Non-secure EL1 execution of the data cache invalidate by set/way instructions to perform a data cache clean and invalidate by set/way.

SWIO	Meaning
0b0	This control has no effect on the operation of data cache invalidate by set/way instructions.
0b1	Data cache invalidate by set/way instructions perform a data cache clean and invalidate by set/way.

When this bit is set to 1, DCISW performs the same invalidation as a DCCISW instruction.

As a result of changes to the behavior of DCISW, this bit is redundant in Armv8. This bit can be implemented as RES1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

VM, bit [0]

Virtualization enable. Enables stage 2 address translation for the Non-secure EL1&0 translation regime.

VM	Meaning
0b0	Non-secure EL1&0 stage 2 address translation disabled.
0b1	Non-secure EL1&0 stage 2 address translation enabled.

If the HCR.DC bit is set to 1, then the behavior of the PE when executing in a Non-secure mode other than Hyp mode is consistent with HCR.VM being 1, regardless of the actual value of HCR.VM, other than the value returned by an explicit read of HCR.VM.

When the value of this bit is 1, data cache invalidate instructions executed at Non-secure EL1 perform a data cache clean and invalidate. For the invalidate by set/way instruction this behavior applies regardless of the value of the HCR.SWIO bit.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HCR2, Hyp Configuration Register 2

The HCR2 characteristics are:

Purpose

Provides additional configuration controls for virtualization.

Configuration

AArch32 System register HCR2 bits [31:0] are architecturally mapped to AArch64 System register HCR_EL2[63:32].

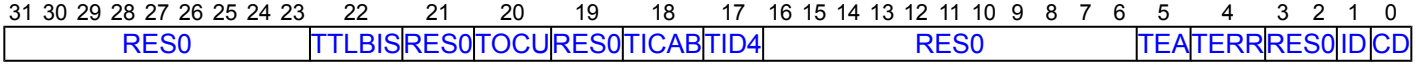
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HCR2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HCR2 is a 32-bit register.

Field descriptions



Bits [31:23]

Reserved, RES0.

TTLBIS, bit [22]

When FEAT_EVT is implemented:

Trap TLB maintenance instructions that operate on the Inner Shareable domain. Traps execution of the following TLB maintenance instructions at EL1 to EL2:

TLBIALLIS, TLBIASIDIS, TLBIMVAAIS, TLBIMVAALIS, TLBIMVAIS, and TLBIMVALIS

TTLBIS	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL1 execution of the specified TLB maintenance instructions is trapped to EL2.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [21]

Reserved, RES0.

TOCU, bit [20]

When FEAT_EVT is implemented:

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of DCCMVAU, ICIALLU, and ICIMVAU at EL1 to EL2.

TOCU	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure execution of the specified cache maintenance instructions is trapped to EL2.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [19]

Reserved, RES0.

TICAB, bit [18]

When FEAT_EVT is implemented:

Trap ICIALLUIS cache maintenance instructions. Traps execution of those cache maintenance instructions at EL1 to EL2.

This applies to the following instructions:

ICIALLUIS.

TICAB	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TID4, bit [17]

When FEAT_EVT is implemented:

Trap ID group 4. Traps the following register accesses to EL2:

- EL1 reads of CCSIDR, CCSIDR2, CLIDR, and CSSELR.
- EL1 writes to CSSELR.

TID4	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified Non-secure EL1 and EL0 accesses to ID group 4 registers are trapped to EL2.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:6]

Reserved, RES0.

TEA, bit [5]

When FEAT_RAS is implemented:

Route synchronous External abort exceptions from EL0 and EL1 to EL2.

TEA	Meaning
0b0	Does not route synchronous External abort exceptions from Non-secure EL0 and EL1 to EL2.
0b1	Route synchronous External abort exceptions from Non-secure EL0 and EL1 to EL2, if not routed to EL3.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TERR, bit [4]

When FEAT_RAS is implemented:

Trap Error record accesses from EL1 to EL2. Traps MRC or MCR accesses, reported using EC syndrome value `0x03`, and MRRC or MCRR accesses, reported using EC syndrome value `0x04`, to the following registers from EL1 to EL2:

ERRIDR, ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXFR, ERXFR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS.

When FEAT_RASv1p1 is implemented, ERXMISC4, ERXMISC5, ERXMISC6, and ERXMISC7.

TERR	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Accesses to the specified registers from EL1 generate a Trap exception to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [3:2]

Reserved, RES0.

ID, bit [1]

Stage 2 Instruction access cacheability disable. For the Non-secure PL1&0 translation regime, when HCR.VM==1, this control forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.

ID	Meaning
`0b0`	This control has no effect on stage 2 of the Non-secure PL1&0 translation regime.
`0b1`	For the Non-secure PL1&0 translation regime, forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.

This bit has no effect on the EL2 translation regime.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CD, bit [0]

Stage 2 Data access cacheability disable. When HCR.VM==1, this forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable for the Non-secure PL1&0 translation regime.

CD	Meaning
`0b0`	This control has no effect on stage 2 of the Non-secure PL1&0 translation regime for data accesses and translation table walks.
`0b1`	For the Non-secure PL1&0 translation regime, forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable.

This bit has no effect on the EL2 translation regime.

The reset behavior of this field is:

- On a Warm reset:
    - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
    - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HCR2

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HCR2;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HCR2;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HCR2 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HCR2 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDCR, Hyp Debug Control Register

The HDCR characteristics are:

Purpose

Controls the trapping to Hyp mode of Non-secure accesses, at EL1 or lower, to functions provided by the debug and trace architectures and the Performance Monitors Extension.

Configuration

AArch32 System register HDCR bits [31:0] are architecturally mapped to AArch64 System register MDCR_EL2[31:0].

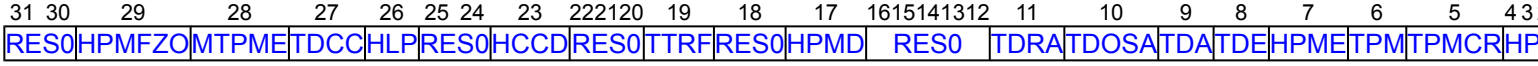
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HDCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3, and other than for a direct read of the register, the PE behaves as if HDCR.HPMN == PMCR.N.

Attributes

HDCR is a 32-bit register.

Field descriptions



Bits [31:30]

Reserved, RES0.

HPMFZO, bit [29]

When FEAT_PMUv3p7 is implemented:

Hyp Performance Monitors Freeze-on-overflow. Stop event counters on overflow.

HPMFZO	Meaning
0b0	Do not freeze on overflow.
0b1	Event counters do not count when PMOVSR[(PMCR.N-1):HDCR.HPMN] is nonzero.

If HDCR.HPMN is less than PMCR.N, this field affects the operation of event counters in the range [HDCR.HPMN .. (PMCR.N-1)].

This field does not affect the operation of other event counters and PMCCNTR.

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTPME, bit [28]

When FEAT_MTPMU is implemented and EL3 is not implemented:

Multi-threaded PMU Enable. Enables use of the PMEVTYPER<n>.MT bits.

MTPME	Meaning
0b0	FEAT_MTPMU is disabled. The Effective value of PMEVTYPER<n>.MT is zero.
0b1	PMEVTYPER<n>.MT bits not affected by this bit.

If FEAT_MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this bit is 0b0.

The reset behavior of this field is:

- On a Cold reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '1'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TDCC, bit [27]

When FEAT_FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel at EL1 and EL0 to EL2.

TDCC	Meaning
0b0	This control does not cause any register accesses to be trapped.
0b1	If EL2 is implemented and enabled in the current Security state, accesses to the DCC registers at EL1 and EL0 generate a Hyp Trap exception, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state.

The DCC registers trapped by this control are:

- DBGDTRRXext, DBGDTRTXext, DBGDSCRint, DBGDCCINT, and, when the PE is in Non-debug state, DBGDTRRXint and DBGDTRTXint.

The traps are reported with EC syndrome value:

- 0x05 for trapped MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped LDC to DBGDTRTXint and STC from DBGDTRRXint.

When the PE is in Debug state, HDCR.TDCC does not trap any accesses to:

- DBGDTRRXint and DBGDTRTXint.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HLP, bit [26]

When FEAT_PMUv3p5 is implemented:

Hypervisor Long event counter enable. Determines when unsigned overflow is recorded by an event counter overflow bit.

HLP	Meaning
0b0	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>[31:0].
0b1	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>[63:0].

If the highest implemented Exception level is using AArch32, it is IMPLEMENTATION DEFINED whether this bit is read/write or RAZ/WI.

If HDCR.HPMN is less than PMCR.N, this bit affects the operation of event counters in the range [HDCR.HPMN..(PMCR.N-1)].

This field does not affect the operation of other event counters.

The operation of this field applies even when EL2 is disabled in the current Security state.

Note

PMEVCNTR<n>[63:32] cannot be accessed directly in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [25:24]

Reserved, RES0.

HCCD, bit [23]

When FEAT_PMUv3p5 is implemented:

Hypervisor Cycle Counter Disable. Prohibits PMCCNTR from counting at EL2.

HCCD	Meaning
0b0	Cycle counting by PMCCNTR is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR is prohibited at EL2.

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [22:20]

Reserved, RES0.

TTRF, bit [19]

When FEAT_TRF is implemented:

Traps use of the Trace Filter Control registers at EL1 to EL2 for MRC or MCR accesses, reported using EC syndrome value `0x03`.

TTRF	Meaning
`0b0`	Accesses to TRFCR at EL1 are not affected by this control bit.
`0b1`	Accesses to TRFCR at EL1 generate a Hyp Trap exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

HPMD, bit [17]

When FEAT_PMUv3p1 is implemented and FEAT_Debugv8p2 is implemented:

Guest Performance Monitors Disable. Controls PMU operation in Hyp mode.

HPMD	Meaning
`0b0`	Counters are not affected by this mechanism.
`0b1`	Affected counters are prohibited from counting in Hyp mode. If PMCR.DP is 1, then PMCCNTR is disabled in Hyp mode. Otherwise, PMCCNTR is not affected by this mechanism.

The counters affected by this field are:

- Event counters PMEVCNTR<n> for values of n less than HDCR.HPMN.
- If PMCR.DP is 1, the cycle counter PMCCNTR.

Other event counters are not affected by this field.

When PMCR.DP is 0, PMCCNTR is not affected by this field.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

When FEAT_PMUv3p1 is implemented:

Guest Performance Monitors Disable. Controls PMU operation in Hyp mode when `ExternalSecureNoninvasiveDebugEnabled()` is FALSE.

HPMD	Meaning
`0b0`	Counters are not affected by this mechanism.
`0b1`	If `ExternalSecureNoninvasiveDebugEnabled()` is FALSE then all the following apply: • Affected event counters are prohibited from counting in Hyp mode. • If PMCR.DP is 1, then PMCCNTR is disabled in Hyp mode. Otherwise, PMCCNTR is not affected by this mechanism.

If `ExternalSecureNoninvasiveDebugEnabled()` is TRUE then the event counters and PMCCNTR are not affected by this field.

Otherwise, the counters affected by this field are:

- Event counters PMEVCNTR<n> for values of n less than HDCR.HPMN.
- If PMCR.DP is 1, the cycle counter, PMCCNTR.

Other event counters are not affected by this field. When PMCR.DP is 0, PMCCNTR is not affected by this field.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:12]

Reserved, RES0.

TDRA, bit [11]

Trap Debug ROM Address register access. Traps Non-secure EL0 and EL1 System register MRC or MCR accesses, reported using EC syndrome value `0x05`, and MRRC accesses, reported using EC syndrome value `0x0C`, to the Debug ROM registers to Hyp mode.

TDRA	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Non-secure EL0 and EL1 System register accesses to the DBGDRAR or DBGDSAR are trapped to Hyp mode, unless it is trapped by DBGDSCRext.UDCCdis.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TDOSA, bit [10]

When FEAT_DoubleLock is implemented:

Trap debug OS-related register access. Traps Non-secure EL1 System register MRC or MCR accesses, reported using EC syndrome value `0x05`, to the powerdown debug registers to Hyp mode.

TDOSA	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Non-secure EL1 System register accesses to the powerdown debug registers are trapped to Hyp mode.

The registers for which accesses are trapped are as follows:

- DBGOSLSR, DBGOSLAR, DBGOSDLR, and DBGPRCR.
- Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

Note

These registers are not accessible at EL0.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trap debug OS-related register access. Traps Non-secure EL1 System register MRC or MCR accesses, reported using EC syndrome value 0x05, to the powerdown debug registers to Hyp mode.

TDOSA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL1 System register accesses to the powerdown debug registers are trapped to Hyp mode.

The registers for which accesses are trapped are as follows:

- DBGOSLSR, DBGOSLAR, and DBGPRCR.
- Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

It is IMPLEMENTATION DEFINED whether accesses to DBGOSDLR are trapped.

Note

These registers are not accessible at EL0.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TDA, bit [9]

Trap debug access. Traps Non-secure EL0 and EL1 System register MRC or MCR accesses, reported using EC syndrome value 0x05, to those debug System registers in the (coproc==0b1110) encoding space that are not trapped by either of the following:

- HDCR.TDRA.
- HDCR.TDOSA.

TDA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure EL0 or EL1 System register accesses to the debug registers, other than the registers trapped by HDCR.TDRA and HDCR.TDOSA, are trapped to Hyp mode, unless it is trapped by DBGDSCRext.UDCCdis.

Traps of AArch32 accesses to DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TDE, bit [8]

Trap Debug exceptions. Controls routing of Debug exceptions, and defines the debug target Exception level, EL_D.

TDE	Meaning
0b0	The debug target Exception level is EL1.
0b1	If EL2 is enabled for the current Effective value of SCR.NS, the debug target Exception level is EL2, otherwise the debug target Exception level is EL1. The HDCR.{TDRA, TDOSA, TDA} fields are treated as being 1 for all purposes other than returning the result of a direct read of the register.

For more information, see 'Routing debug exceptions'.

When HCR.TGE == 1, the PE behaves as if the value of this field is 1 for all purposes other than returning the value of a direct read of the register.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

HPME, bit [7]
When FEAT_PMUv3 is implemented:

Hyp Enable.

HPME	Meaning
0b0	Affected counters are disabled and do not count.
0b1	Affected counters are enabled by PMCNTENSET.

The counters affected by this field are event counters PMEVCNTR<n> for values of n greater than or equal to HDCR.HPMN and less than PMCR.N. This applies even when EL2 is disabled in the current Security state.

Other event counters and PMCCNTR are not affected by this field.

If HDCR.HPMN is equal to PMCR.N, then this field has no effect.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPM, bit [6]
When FEAT_PMUv3 is implemented:

Trap accesses of PMU registers. Enables a trap to EL2 on accesses of PMU registers.

TPM	Meaning
0b0	Accesses of the specified PMU registers are not trapped by this mechanism.
0b1	Accesses of the specified PMU registers at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.

The instructions affected by this control are:

- MRC and MCR accesses to PMCCFILTR, PMCCNTR, PMCNTENCLR, PMCNTENSET, PMCR, PMEVCNTR<n>, PMEVTYPER<n>, PMINTENCLR, PMINTENSET, PMOVSR, PMOVSSET, PMSELR, PMSWINC, PMUSERENR, PMXEVCNTR, and PMXEVTYPER.
- MRC accesses to PMCEID0 and PMCEID1.
- MRRC and MCRR accesses to PMCCNTR.
- If FEAT_PMUv3p1 is implemented, MRC accesses to PMCEID2 and PMCEID3.
- If FEAT_PMUv3p4 is implemented, MRC accesses to PMMIR.

Unless the instruction generates a higher priority exception, trapped instructions generate a Hyp Trap exception.

Trapped instructions are reported using EC syndrome value `0x03` for `MRC` and `MCR` accesses, and `0x04` for `MRRC` and `MCRR` accesses.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPMCR, bit [5]

When FEAT_PMUv3 is implemented:

Trap PMCR accesses. Traps Non-secure EL0 and EL1 MCR or MRC accesses to the PMCR to Hyp mode, reported using EC syndrome value `0x03`.

TPMCR	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Non-secure EL0 and EL1 accesses to the PMCR are trapped to Hyp mode, unless it is trapped by PMUSERENR.EN.

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPMN, bits [4:0]

When FEAT_PMUv3 is implemented:

Defines the number of event counters PMEVCNTR<n> that are accessible from EL1 and, if permitted, from EL0.

HDCR.HPMN divides the event counters into a first range and a second range.

If HDCR.HPMN is not 0 and is less than the number of PMU event counters implemented by the PE, NUM_PMU_COUNTERS, then event counters [0..(HDCR.HPMN-1)] are in the first range, and the remaining event counters [HDCR.HPMN..(NUM_PMU_COUNTERS-1)] are in the second range.

If FEAT_HPMN0 is implemented and HDCR.HPMN is 0, then all of the following apply:

- No event counters are in the first range.
- All event counters are in the second range.

If HDCR.HPMN is equal to NUM_PMU_COUNTERS, or EL2 is not implemented, then all of the following apply:

- All event counters are in the first range.
- No event counters are in the second range.

All of the following apply for an event counter PMEVCNTR<n> in the first range:

- The counter is accessible from EL1, EL2, and EL3.
- The counter is accessible from EL0 if permitted by PMUSERENR.
- If FEAT_PMUv3p5 is implemented, then PMCR.LP determines whether the counter overflow flag PMOVSSET[n] is set on unsigned overflow of PMEVCNTR<n>[31:0] or PMEVCNTR<n>[63:0]. PMEVCNTR<n>[63:32] cannot be accessed directly in AArch32 state.
- PMCR.E and PMCNTENSET[n] enable the operation of the event counter.

All of the following apply for an event counter PMEVCNTR<n> in the second range:

- The counter is accessible from EL2 and EL3.
- If EL2 is disabled in the current Security state, then the event counter is accessible from EL1, and from EL0 if permitted by PMUSERENR.
- If FEAT_PMUv3p5 is implemented, HDCR.HLP determines whether the counter overflow flag PMOVSSET[n] is set on unsigned overflow of PMEVCNTR<n>[31:0] or PMEVCNTR<n>[63:0]. PMEVCNTR<n>[63:32] cannot be accessed directly in AArch32 state.
- HDCR.HPME and PMCNTENSET[n] enable the operation of the event counter.

Values greater than NUM_PMU_COUNTERS are reserved. If FEAT_HPMN0 is not implemented then the value 0 is reserved.

If this field is set to a reserved value, then the following CONSTRAINED UNPREDICTABLE behaviors apply:

- The value returned by a direct read of HDCR.HPMN is UNKNOWN.
- The number of event counters in each of the first and second ranges is UNKNOWN. That is, either:
  - The PE behaves as if HDCR.HPMN is set to an UNKNOWN nonzero value less than or equal to NUM_PMU_COUNTERS.
  - All counters are in the second range and none are in the first range.

The reset behavior of this field is:

- On a Warm reset, this field resets to the expression `NUM_PMU_COUNTERS`.

Otherwise:

Reserved, RES0.

Accessing HDCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = HDCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HDCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        HDCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HDCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDFAR, Hyp Data Fault Address Register

The HDFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Data Abort exception that is taken to Hyp mode.

Configuration

AArch32 System register HDFAR bits [31:0] are architecturally mapped to AArch64 System register FAR_EL2[31:0].

AArch32 System register HDFAR bits [31:0] are architecturally mapped to AArch32 System register DFAR[31:0] (DFAR_S) when FEAT_AA32EL2 is implemented and FEAT_AA32EL3 is implemented.

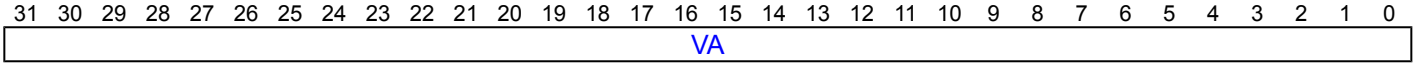
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HDFAR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HDFAR is a 32-bit register.

Field descriptions



VA, bits [31:0]

VA of faulting address of synchronous Data Abort exception taken to Hyp mode.

On a Prefetch Abort exception, this register is UNKNOWN.

Any execution in a Non-secure EL1 or Non-secure EL0 mode makes this register UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HDFAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T6
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T6 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HDFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HDFAR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T6
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T6 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HDFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HDFAR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HIFAR, Hyp Instruction Fault Address Register

The HIFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Prefetch Abort exception that is taken to Hyp mode.

Configuration

AArch32 System register HIFAR bits [31:0] are architecturally mapped to AArch64 System register FAR_EL2[63:32].

AArch32 System register HIFAR bits [31:0] are architecturally mapped to AArch32 System register IFAR[31:0] (IFAR_S) when EL2 is implemented and EL3 is implemented.

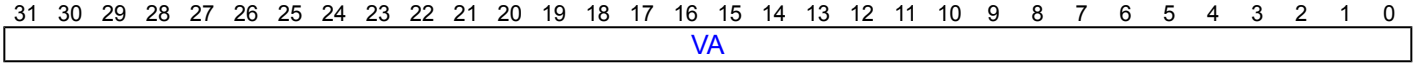
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HIFAR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HIFAR is a 32-bit register.

Field descriptions



VA, bits [31:0]

VA of faulting address of synchronous Prefetch Abort exception taken to Hyp mode.

On a Data Abort exception, this register is UNKNOWN.

Any execution in a Non-secure EL1 or Non-secure EL0 mode makes this register UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HIFAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0110	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T6
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T6 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HIFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HIFAR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0110	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T6
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T6 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HIFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HIFAR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HMAIR0, Hyp Memory Attribute Indirection Register 0

The HMAIR0 characteristics are:

Purpose

Along with HMAIR1, provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations for memory accesses from Hyp mode.

AttrIndx[2] indicates the HMAIR register to be used:

- When AttrIndx[2] is 0, HMAIR0 is used.
- When AttrIndx[2] is 1, HMAIR1 is used.

Configuration

AArch32 System register HMAIR0 bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL2[31:0].

This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HMAIR0 are UNDEFINED.

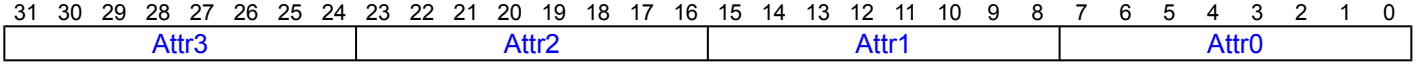
If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HMAIR0 is a 32-bit register.

Field descriptions

When TTBCR.EAE == 1:



Attr<n>, bits [8n+7:8n], for n = 3 to 0

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIndx[2:0] gives the value of <n> in Attr<n>.
- AttrIndx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

Attr<n>[7:4]	Meaning
0b0000	Device memory. See encoding of Attr<n>[3:0] for the type of Device memory.
0b00RW, RW not 0b00	Normal memory, Outer Write-Through Transient.
0b0100	Normal memory, Outer Non-cacheable.
0b01RW, RW not 0b00	Normal memory, Outer Write-Back Transient.
0b10RW	Normal memory, Outer Write-Through Non-transient.
0b11RW	Normal memory, Outer Write-Back Non-transient.

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

Attr<n>[3:0]	Meaning when Attr<n>[7:4] is 0b0000	Meaning when Attr<n>[7:4] is not 0b0000
0b0000	Device-nGnRnE memory	UNPREDICTABLE
0b00RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Through Transient
0b0100	Device-nGnRE memory	Normal memory, Inner Non-cacheable
0b01RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Back Transient
0b1000	Device-nGRE memory	Normal memory, Inner Write-Through Non-transient (RW=0b00)
0b10RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Through Non-transient
0b1100	Device-GRE memory	Normal memory, Inner Write-Back Non-transient (RW=0b00)
0b11RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Back Non-transient

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<n> fields have the following meanings:

R or W	Meaning
0b0	No Allocate
0b1	Allocate

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HMAIR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HMAIR0;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HMAIR0;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HMAIR0 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HMAIR0 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HMAIR1, Hyp Memory Attribute Indirection Register 1

The HMAIR1 characteristics are:

Purpose

Along with HMAIR0, provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations for memory accesses from Hyp mode.

AttrIndx[2] indicates the HMAIR register to be used:

- When AttrIndx[2] is 0, HMAIR0 is used.
- When AttrIndx[2] is 1, HMAIR1 is used.

Configuration

AArch32 System register HMAIR1 bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL2[63:32].

This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HMAIR1 are UNDEFINED.

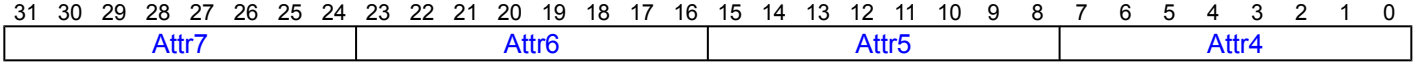
If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HMAIR1 is a 32-bit register.

Field descriptions

When TTBCR.EAE == 1:



Attr<n>, bits [8(n-4)+7:8(n-4)], for n = 7 to 4

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIndx[2:0] gives the value of <n> in Attr<n>.
- AttrIndx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

Attr<n>[7:4]	Meaning
0b0000	Device memory. See encoding of Attr<n>[3:0] for the type of Device memory.
0b00RW, RW not 0b00	Normal memory, Outer Write-Through Transient.
0b0100	Normal memory, Outer Non-cacheable.
0b01RW, RW not 0b00	Normal memory, Outer Write-Back Transient.
0b10RW	Normal memory, Outer Write-Through Non-transient.
0b11RW	Normal memory, Outer Write-Back Non-transient.

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

Attr<n>[3:0]	Meaning when Attr<n>[7:4] is 0b0000	Meaning when Attr<n>[7:4] is not 0b0000
0b0000	Device-nGnRnE memory	UNPREDICTABLE
0b00RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Through Transient
0b0100	Device-nGnRE memory	Normal memory, Inner Non-cacheable
0b01RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Back Transient
0b1000	Device-nGRE memory	Normal memory, Inner Write-Through Non-transient (RW=0b00)
0b10RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Through Non-transient
0b1100	Device-GRE memory	Normal memory, Inner Write-Back Non-transient (RW=0b00)
0b11RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Back Non-transient

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<n> fields have the following meanings:

R or W	Meaning
0b0	No Allocate
0b1	Allocate

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HMAIR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1010	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HMAIR1;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HMAIR1;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1010	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HMAIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HMAIR1 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HPFAR, Hyp IPA Fault Address Register

The HPFAR characteristics are:

Purpose

Holds the faulting IPA for some aborts on a stage 2 translation taken to Hyp mode.

Configuration

AArch32 System register HPFAR bits [31:0] are architecturally mapped to AArch64 System register HPFAR_EL2[31:0].

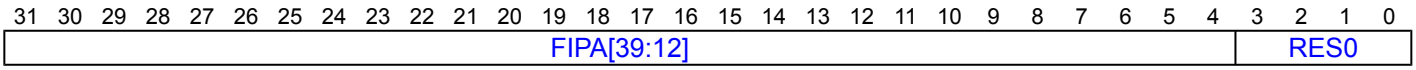
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HPFAR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HPFAR is a 32-bit register.

Field descriptions



Execution in any Non-secure mode other than Hyp mode makes this register UNKNOWN.

FIPA[39:12], bits [31:4]

Bits [39:12] of the faulting intermediate physical address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:0]

Reserved, RES0.

Accessing HPFAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0110	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T6
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T6 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HPFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HPFAR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0110	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T6
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T6 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HPFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HPFAR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HRMR, Hyp Reset Management Register

The HRMR characteristics are:

Purpose

If EL2 is the highest implemented Exception level and this register is implemented:

- A write to the register at EL2 can request a Warm reset.
- If EL2 can use AArch32 and AArch64, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch32 System register HRMR bits [31:0] are architecturally mapped to AArch64 System register RMR_EL2[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to HRMR are UNDEFINED.

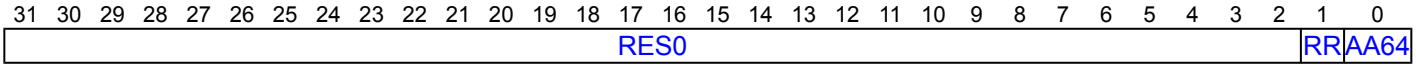
Only implemented if EL2 is the highest implemented Exception level. In this case:

- If EL2 can use AArch32 and AArch64 then this register must be implemented.
- If EL2 cannot use AArch64 then it is IMPLEMENTATION DEFINED whether the register is implemented.

Attributes

HRMR is a 32-bit register.

Field descriptions



Bits [31:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

AA64, bit [0]

When EL2 can use AArch64, determines which Execution state the PE boots into after a Warm reset:

AA64	Meaning
0b0	AArch32.
0b1	AArch64.

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL2 cannot use AArch64 this bit is RAZ/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing HRMR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2)
&& !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2)
&& ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    R[t] = HRMR;
else
    UNDEFINED;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2)
&& !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2)
&& ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    HRMR = R[t];
else
    UNDEFINED;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HSCTLR, Hyp System Control Register

The HSCTLR characteristics are:

Purpose

Provides top-level control of the system operation in Hyp mode.

Configuration

AArch32 System register HSCTLR bits [31:0] are architecturally mapped to AArch64 System register SCTLR_EL2[31:0].

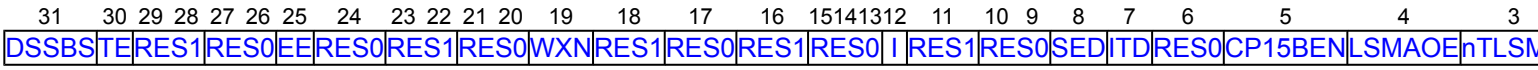
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HSCTLR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HSCTLR is a 32-bit register.

Field descriptions



DSSBS, bit [31]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry. The defined values are:

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to Hyp mode.
0b1	PSTATE.SSBS is set to 1 on an exception to Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

TE, bit [30]

T32 Exception Enable. This bit controls whether exceptions to EL2 are taken to A32 or T32 state:

TE	Meaning
0b0	Exceptions, including reset, taken to A32 state.
0b1	Exceptions, including reset, taken to T32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [29:28]

Reserved, RES1.

Bits [27:26]

Reserved, RES0.

EE, bit [25]

The value of the PSTATE.E bit on entry to Hyp mode, the endianness of stage 1 translation table walks in the EL2 translation regime, and the endianness of stage 2 translation table walks in the PL1&0 translation regime.

EE	Meaning
0b0	Little-endian. PSTATE.E is cleared to 0 on entry to Hyp mode. Stage 1 translation table walks in the EL2 translation regime, and stage 2 translation table walks in the PL1&0 translation regime are little-endian.
0b1	Big-endian. PSTATE.E is set to 1 on entry to Hyp mode. Stage 1 translation table walks in the EL2 translation regime, and stage 2 translation table walks in the PL1&0 translation regime are big-endian.

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bit [24]

Reserved, RES0.

Bits [23:22]

Reserved, RES1.

Bits [21:20]

Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL2 translation regime, this bit can force all memory regions that are writable to be treated as XN.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the EL2 translation regime is forced to XN for accesses from software executing at EL2.

This bit applies only when HSCTLR.M bit is set.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [18]

Reserved, RES1.

Bit [17]

Reserved, RES0.

Bit [16]

Reserved, RES1.

Bits [15:13]

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL2:

I	Meaning
0b0	All instruction access to Normal memory from EL2 are Non-cacheable for all levels of instruction and unified cache. If the value of HSCTLR.M is 0, instruction accesses from stage 1 of the EL2 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	All instruction access to Normal memory from EL2 can be cached at all levels of instruction and unified cache. If the value of HSCTLR.M is 0, instruction accesses from stage 1 of the EL2 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

This bit has no effect on the PL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES1.

Bits [10:9]

Reserved, RES0.

SED, bit [8]

SETEND instruction disable. Disables SETEND instructions at EL2.

SED	Meaning
0b0	SETEND instruction execution is enabled at EL2.
0b1	SETEND instructions are UNDEFINED at EL2.

If the implementation does not support mixed-endian operation at EL2, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ITD, bit [7]

IT Disable. Disables some uses of IT instructions at EL2.

ITD	Meaning
0b0	All IT instruction functionality is enabled at EL2.
0b1	Any attempt at EL2 to execute any of the following is UNDEFINED: • All encodings of the IT instruction with hw1[3:0]!=1000. • All encodings of the subsequent instruction with the following values for hw1: ◦ 0b11xxxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. ◦ 0b1011xxxxxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions'. ◦ 0b10100xxxxxxxxxx: ADD Rd, PC, #imm ◦ 0b01001xxxxxxxxxx: LDR Rd, [PC, #imm] ◦ 0b0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. ◦ 0b010001xx1xxxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn. These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block. It is IMPLEMENTATION DEFINED whether the IT instruction is treated as: • A 16-bit instruction, that can only be followed by another 16-bit instruction. • The first half of a 32-bit instruction. This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED. An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information, see 'Changes to an ITD control by an instruction in an IT block'.

ITD is optional, but if it is implemented in the HSCTLR then it must also be implemented in the SCTLR_EL1, SCTLR_EL2, and SCTLR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement ITD, access to this field is **RAZ/WI**.

Bit [6]

Reserved, RES0.

CP15BEN, bit [5]

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL2:

CP15BEN	Meaning
0b0	EL2 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED.
0b1	EL2 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

CP15BEN is optional, but if it is implemented in the HSCTLR then it must also be implemented in the SCTLR_EL1, SCTLR_EL2, and SCTLR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement CP15BEN, access to this field is **RAO/WI**.

LSMAOE, bit [4]

When FEAT_LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

LSMAOE	Meaning
0b0	For all memory accesses at EL2, T32 and A32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1	The ordering and interrupt behavior of T32 and A32 Load Multiple and Store Multiple at EL2 is as defined for Armv8.0.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '1'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

nTLSMD, bit [3]

When FEAT_LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

nTLSMD	Meaning
0b0	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL2 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL2 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '1'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

C, bit [2]

Cacheability control, for data accesses at EL2:

C	Meaning
0b0	All data access to Normal memory from EL2, and all accesses to the EL2 translation tables, are Non-cacheable for all levels of data and unified cache.
0b1	All data access to Normal memory from EL2, and all accesses to the EL2 translation tables, can be cached at all levels of data and unified cache.

This bit has no effect on the PL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL2:

A	Meaning
0b0	Alignment fault checking disabled when executing at EL2. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element or data elements being accessed.
0b1	Alignment fault checking enabled when executing at EL2. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element or data elements being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL2 stage 1 address translation. Possible values of this bit are:

M	Meaning
0b0	EL2 stage 1 address translation disabled. See the HSCTLR.I field for the behavior of instruction accesses to Normal memory.
0b1	EL2 stage 1 address translation enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HSCTLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HSCTLR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HSCTLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HSCTLR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HSCTLR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HSR, Hyp Syndrome Register

The HSR characteristics are:

Purpose

Holds syndrome information for an exception taken to Hyp mode.

Configuration

AArch32 System register HSR bits [31:0] are architecturally mapped to AArch64 System register ESR_EL2[31:0].

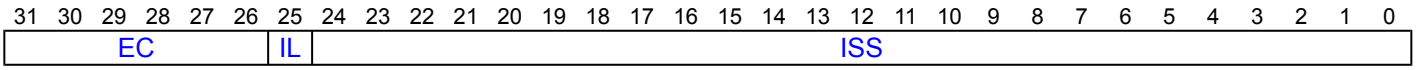
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HSR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HSR is a 32-bit register.

Field descriptions



Execution in any Non-secure PE mode other than Hyp mode makes this register UNKNOWN.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL2, the value of HSR is UNKNOWN. The value written to HSR must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

EC, bits [31:26]

Exception Class. Indicates the reason for the exception that this register holds information about. Possible values of this field are:

EC	Meaning	ISS
0b000000	Unknown reason.	ISS encoding for exceptions with an unknown reason
0b000001	Trapped WFI or WFE instruction execution. Conditional WFE and WFI instructions that fail their condition code check do not cause an exception.	ISS encoding for Exception from a WFI or WFE instruction
0b000011	Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC value 0b000000.	ISS encoding for Exception from an MCR or MRC access
0b000100	Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC value 0b000000.	ISS encoding for Exception from an MCRR or MRRC access
0b000101	Trapped MCR or MRC access with (coproc==0b1110).	ISS encoding for Exception from an MCR or MRC access
0b000110	Trapped LDC or STC access. The only architected uses of these instructions are: • An STC to write data to memory from DBGDTRRXint. • An LDC to read data from memory to DBGDTRTXint.	ISS encoding for Exception from an LDC or STC instruction
0b000111	Access to Advanced SIMD or floating-point functionality trapped by a HCPTR.{TASE, TCP10} control. Excludes exceptions generated because Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000.	ISS encoding for Exception from an access to SIMD or floating-point functionality, resulting from HCPTR
0b001000	Trapped VMRS access, from ID group trap, that is not reported using EC value 0b000111.	ISS encoding for Exception from an MCR or MRC access
0b001100	Trapped MRRC access with (coproc==0b1110).	ISS encoding for Exception from an MCRR or MRRC access
0b001110	Illegal exception return to AArch32 state.	ISS encoding for Exception from an Illegal state or PC alignment fault
0b010001	Exception on SVC instruction execution in AArch32 state routed to EL2.	ISS encoding for Exception from HVC or SVC instruction execution
0b010010	HVC instruction execution in AArch32 state, when HVC is not disabled.	ISS encoding for Exception from HVC or SVC instruction execution
0b010011	Trapped execution of SMC instruction in AArch32 state.	ISS encoding for Exception from SMC instruction execution
0b100000	Prefetch Abort from a lower Exception level.	ISS encoding for Exception from a Prefetch Abort
0b100001	Prefetch Abort taken without a change in Exception level.	ISS encoding for Exception from a Prefetch Abort
0b100010	PC alignment fault exception.	ISS encoding for Exception from an Illegal state or PC alignment fault
0b100100	Data Abort exception from a lower Exception level.	ISS encoding for Exception from a Data Abort
0b100101	Data Abort exception taken without a change in Exception level.	ISS encoding for Exception from a Data Abort

All other EC values are reserved by Arm, and:

- Unused values in the range 0b000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
- Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [25]

Instruction length bit. Indicates the size of the instruction that has been trapped to Hyp mode. When this bit is valid, possible values of this bit are:

IL	Meaning
0b0	16-bit instruction trapped.
0b1	32-bit instruction trapped.

This field is RES1 and not valid for the following cases:

- When the EC value is 0b000000, indicating an exception with an unknown reason.
- Prefetch Aborts.
- Data Abort exceptions for which the HSR.ISS.ISV field is 0.
- When the EC value is 0b001110, indicating an Illegal state exception.

The IL field is not valid and is UNKNOWN on an exception from a PC alignment fault.

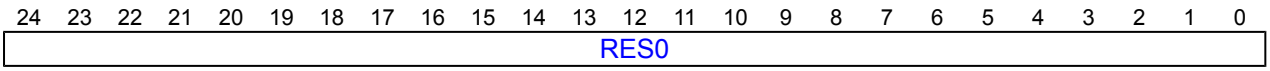
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [24:0]

Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

ISS encoding for exceptions with an unknown reason



Bits [24:0]

Reserved, RES0.

Additional information for the ISS encoding for exceptions with an unknown reason

This EC value is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or is not accessible in the current PE mode in the current Security state, including:
  - A read access using a System register encoding pattern that is not allocated for reads or that does not permit reads in the current PE mode and Security state.
  - A write access using a System register encoding pattern that is not allocated for writes or that does not permit writes in the current PE mode and Security state.
  - Instruction encodings that are unallocated.
  - Instruction encodings for instructions not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
- The attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the SCTLR.{ITD, SED, CP15BEN} control bits.
- Attempted execution of:
  - An HVC instruction when disabled by HCR.HCD, SCR.HCE, or SCR_EL3.HCE.
  - An SMC instruction when disabled by SCR.SCD or SCR_EL3.SMD.
  - An HLT instruction when disabled by EDSCR.HDE.
- An HVC instruction when disabled by HCR.HCD, SCR.HCE, or SCR_EL3.HCE.An SMC instruction when disabled by SCR.SCD or SCR_EL3.SMD.An HLT instruction when disabled by EDSCR.HDE.

- An exception generated because of the attempted execution of an MSR (Banked register) or MRS (Banked register) instruction that would access a Banked register that is not accessible from the Security state and PE mode at which the instruction was executed.

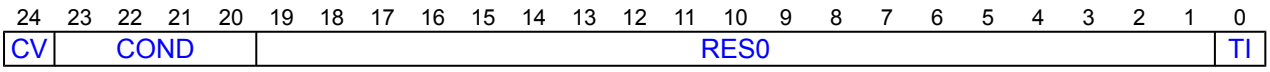
Note

An exception is generated only if the CONSTRAINED UNPREDICTABLE behavior of the instruction is that it is UNDEFINED, see 'MSR (banked register) and MRS (banked register)'.

- Attempted execution, in Debug state, of:
  - A DCPS1 instruction in Non-secure state from EL0 when EL2 is using AArch32 and the value of HCR.TGE is 1.
  - A DCPS2 instruction at EL1 or EL0 when EL2 is not implemented, or when EL3 is using AArch32 and the value of SCR.NS is 0, or when EL3 is using AArch64 and the value of SCR_EL3.NS is 0.
  - A DCPS3 instruction when EL3 is not implemented, or when the value of EDSCR.SDD is 1.
- In Debug state when the value of EDSCR.SDD is 1, the attempted execution at EL2, EL1, and EL0 of an instruction that is configured to trap to EL3.

'Undefined Instruction exception, when the value of HCR.TGE is 1' describes the configuration settings for a trap that returns an EC value of `0b000000`.

ISS encoding for Exception from a WFI or WFE instruction



CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to `0b1110`.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to `0b1110`, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:1]

Reserved, RES0.

TI, bit [0]

Trapped instruction. Possible values of this bit are:

TI	Meaning
0b0	WFI trapped.
0b1	WFE trapped.

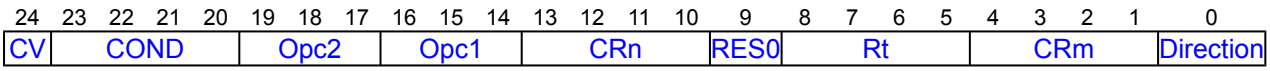
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for Exception from a WFI or WFE instruction

HCR.{TWE, TWI} describe the configuration settings for this trap.

ISS encoding for Exception from an MCR or MRC access



CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc2, bits [19:17]

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value `0b000`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [16:14]

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value `0b111`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

Rt, bits [8:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value `0b0000`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write to System register space. MCR instruction.
0b1	Read from System register space. MRC or VMRS instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for Exception from an MCR or MRC access

The following fields describe configuration settings for traps from an MCR or MCR access using coproc 0b1111 that are reported using EC value 0b000011:

- HCR.{TID1, TID2, TID3}, for Non-secure accesses to the ID registers at EL0 and EL1, trapped to EL2.
- HCR.TIDCP, for Non-secure accesses to lockdown, DMA, and TCM operations at EL0 and EL1, trapped to EL2.
- HCR.{TSW, TPC, TPU}, for Non-secure execution of cache maintenance instructions at EL1, trapped to EL2.
- HCR.TTLB, for Non-secure execution of TLB maintenance instructions at EL1, trapped to EL2.
- HCR.TAC, for Non-secure accesses to the Auxiliary Control Register at EL1, trapped to EL2.
- HDCR.{TPM, TPMCR}, for Non-secure accesses to Performance Monitors registers at EL0 and EL1, trapped to EL2.
- HCPTR.TAM, for Non-secure accesses to Activity Monitor registers at EL0 and EL1, trapped to EL2.
- HCPTR.TCPAC, for Non-secure accesses to the CPACR at EL1, trapped to EL2.
- HCR.{TRVM, TVM}, for Non-secure accesses to virtual memory control registers at EL1, trapped to EL2.
- HSTR.T<n>, for Non-secure accesses to System registers in the (coproc == 1111) encoding space at EL0 and EL1, trapped to EL2.
- HDCR.TTRF, for Non-secure accesses to trace filter control registers from System register, trapped to EL2.
- CNTHCTL.PL1PCEN, for Non-secure accesses to the Generic Timer registers at EL0 and EL1, trapped to EL2.
- HCR2.TERR, for Non-secure accesses to the RAS error record registers at EL1, trapped to EL2.

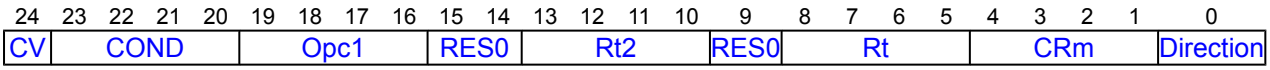
The following fields describe configuration settings for traps from an MCR or MRC access using coproc 0b1110 that are reported using EC value 0b000101:

- HCR.TID0, for Non-secure accesses to the Primary device identification registers at EL0 and EL1, trapped to EL2.
- HCPTR.TTA, for Non-secure accesses to trace registers from System register, trapped to EL2.
- HDCR.TDRA, for Non-secure accesses to Debug ROM registers from System register, trapped to EL2.
- HDCR.TDOSA, for Non-secure accesses to powerdown debug registers from System register, trapped to EL2.
- HDCR.TDA, for Non-secure accesses to debug registers from System register, trapped to EL2.

The following fields describes configuration settings for traps from a VMSR or VMRS access that are reported using EC value 0b001000:

- HCR.TID0, for Non-secure accesses to the Primary device identification registers at EL1, for ID group traps trapped to EL2.
- HCR.TID3, for Non-secure accesses to the Detailed feature identification registers at EL0 and EL1, for ID group traps trapped to EL2.
- HCPTR.{TCP10, TCP11}, for Non-secure accesses to FPSCR, FPSID, FPEXC, MVFR0, MVFR1, and MVFR2, trapped to EL2.

ISS encoding for Exception from an MCRR or MRRC access



CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to `0b1110`.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to `0b1110`, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [19:16]

The Opc1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:14]

Reserved, RES0.

Rt2, bits [13:10]

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

Rt, bits [8:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write to System register space. MCRR instruction.
0b1	Read from System register space. MRRC instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for Exception from an MCRR or MRRC access

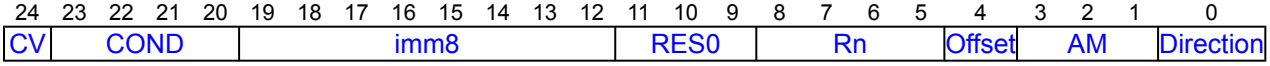
The following fields describe configuration settings for traps from an MCRR or MRRC access using coproc 0b1111 that are reported using EC value 0b000100:

- HCR.{TRVM, TVM}, for Non-secure accesses to virtual memory control registers at EL1, trapped to EL2.
- HDCR.TPM, for Non-secure accesses to Performance Monitors registers at EL0 and EL1, trapped to EL2.
- HCPTR.TAM, for Non-secure accesses to Activity Monitor registers at EL0 and EL1, trapped to EL2.
- CNTHCTL.{PL1PCEN, PL1PCTEN}, for Non-secure accesses to the Generic Timer registers at EL0 and EL1, trapped to EL2.
- HSTR.T<n>, for Non-secure accesses to System registers in the (coproc == 1111) encoding space at EL0 and EL1, trapped to EL2.
- HCR2.TERR, for Non-secure accesses to the RAS error record registers at EL1, trapped to EL2.

The following fields describe configuration settings for traps from an MCRR or MRRC access using coproc 0b1110 that are reported using EC value 0b001100:

- HCPTR.TTA, for Non-secure accesses to trace registers from System register, trapped to EL2.
- HDCR.TDRA, for Non-secure accesses to Debug ROM registers from System register, trapped to EL2.

ISS encoding for Exception from an LDC or STC instruction



CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to `0b1110`, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

imm8, bits [19:12]

The immediate value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:9]

Reserved, RES0.

Rn, bits [8:5]

The Rn value from the issued instruction. Valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction.

When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Offset, bit [4]

Indicates whether the offset is added or subtracted:

Offset	Meaning
`0b0`	Subtract offset.
`0b1`	Add offset.

This bit corresponds to the U bit in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AM, bits [3:1]

Addressing mode. The permitted values of this field are:

AM	Meaning
0b000	Immediate unindexed.
0b001	Immediate post-indexed.
0b010	Immediate offset.
0b011	Immediate pre-indexed.
0b100	Literal unindexed. LDC instruction in A32 instruction set only. For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved.
0b110	Literal offset. LDC instruction only. For a trapped STC instruction, this encoding is reserved.

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

Direction	Meaning
0b0	Write to memory. STC instruction.
0b1	Read from memory. LDC instruction.

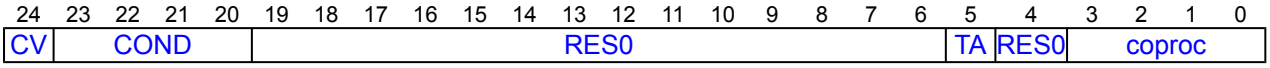
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for Exception from an LDC or STC instruction

HDCR.TDA describes the configuration settings for the trap that is reported using EC value 0b000110.

ISS encoding for Exception from an access to SIMD or floating-point functionality, resulting from HCPTR



Excludes exceptions that occur because Advanced SIMD and floating-point functionality is not implemented, or because the value of HCR.TGE or HCR_EL2.TGE is 1. These are reported with EC value 0b000000.

CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
0b0	The COND field is not valid.
0b1	The COND field is valid.

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to `0b1110`.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to `0b1110`, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:6]

Reserved, RES0.

TA, bit [5]

Indicates trapped use of Advanced SIMD functionality.

TA	Meaning
`0b0`	Exception was not caused by trapped use of Advanced SIMD functionality.
`0b1`	Exception was caused by trapped use of Advanced SIMD functionality.

Any use of an Advanced SIMD instruction that is not also a floating-point instruction that is trapped to Hyp mode because of a trap configured in the HCPTR sets this bit to 1.

For a list of these instructions, see 'Controls of Advanced SIMD operation that do not apply to floating-point operation'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [4]

Reserved, RES0.

coproc, bits [3:0]

When the HSR.TA field returns the value 1, this field returns the value `0b1010`. Otherwise, this field is RES0.

The reset behavior of this field is:

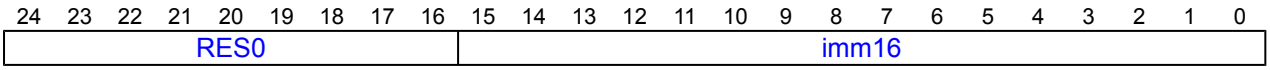
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for Exception from an access to SIMD or floating-point functionality, resulting from HCPTR

The following fields describe the configuration settings for the traps that are reported using EC value `0b000111`:

- HCPTR.{TCP11, TCP10}, for Non-secure accesses to the SIMD and floating-point registers, trapped to EL2.
- HCPTR.TASE, for Non-secure accesses to Advanced SIMD functionality, trapped to EL2.

ISS encoding for Exception from HVC or SVC instruction execution



Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, this is the value of the imm16 field of the issued instruction.

For an SVC instruction:

- If the instruction is unconditional, then:
  - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
  - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

The reset behavior of this field is:

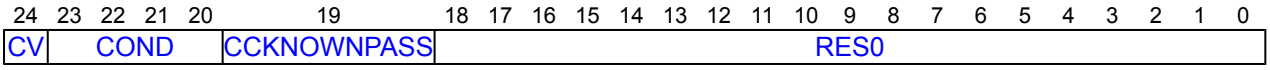
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for Exception from HVC or SVC instruction execution

The HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.

'Supervisor Call exception, when the value of HCR.TGE is 1' describes the configuration settings for the trap reported with EC value `0b010001`.

ISS encoding for Exception from SMC instruction execution



CV, bit [24]

Condition code valid. Possible values of this bit are:

CV	Meaning
`0b0`	The COND field is not valid.
`0b1`	The COND field is valid.

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to `0b1110`.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to `0b1110`, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both T32 and A32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to `0b1110`, or to the value of any condition that applied to the instruction.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CCKNOWNPASS, bit [19]

Indicates whether the instruction might have failed its condition code check.

CCKNOWNPASS	Meaning
`0b0`	The instruction was unconditional, or was conditional and passed its condition code check.
`0b1`	The instruction was conditional, and might have failed its condition code check.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

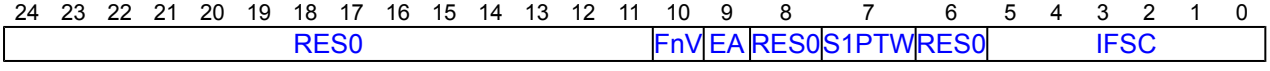
Bits [18:0]

Reserved, RES0.

Additional information for the ISS encoding for Exception from SMC instruction execution

HCR.TSC describes the configuration settings for this trap for instructions executed in Non-secure EL1.

ISS encoding for Exception from a Prefetch Abort



Bits [24:11]

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	HIFAR is valid.
0b1	HIFAR is not valid, and holds an UNKNOWN value.

This field is valid only if the IFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code. Possible values of this field are:

IFSC	Meaning	Applies when
0b000000	Address size fault in translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk.	
0b010101	Synchronous External abort on translation table walk, level 1.	
0b010110	Synchronous External abort on translation table walk, level 2.	
0b010111	Synchronous External abort on translation table walk, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk, level 3.	When FEAT_RAS is not implemented
0b100010	Debug exception.	
0b110000	TLB conflict abort.	

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

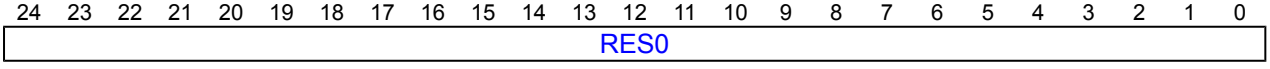
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for Exception from a Prefetch Abort

The following sections describe cases where Prefetch Abort exceptions can be routed to Hyp mode, generating exceptions that are reported in the HSR with EC value 0b100000:

- 'Abort exceptions, when the value of HCR.TGE is 1'.
- 'Routing debug exceptions to EL2 using AArch32'.

ISS encoding for Exception from an Illegal state or PC alignment fault



Bits [24:0]

Reserved, RES0.

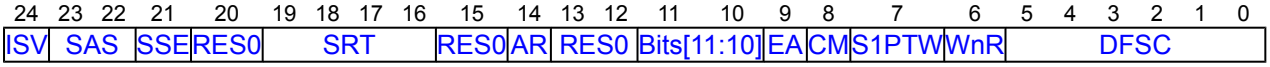
Additional information for the ISS encoding for Exception from an Illegal state or PC alignment fault

For more information about the Illegal state exception, see:

- 'Illegal changes to PSTATE.M'.
- 'Illegal return events from AArch32 state'.
- 'Legal returns that set PSTATE.IL to 1'.
- 'The Illegal Execution state exception'.

For more information about the PC alignment fault exception, see 'Branching to an unaligned PC'.

ISS encoding for Exception from a Data Abort



ISV, bit [24]

Instruction Syndrome Valid. Indicates whether the syndrome information in ISS[23:14] is valid.

ISV	Meaning
0b0	No valid instruction syndrome. ISS[23:14] are RES0.
0b1	ISS[23:14] hold a valid instruction syndrome.

This bit is 0 for all faults except Data Abort exceptions generated by stage 2 address translations for which all the following apply to the instruction that generated the Data Abort exception:

- The instruction is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
- The instruction is not performing register writeback.
- The instruction is not using the PC as a source or destination register.

For these cases, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, as described in 'Data Abort exceptions in Memory access mode', and otherwise indicates whether ISS[23:14] hold a valid syndrome.

Note

In the A32 instruction set, LDR*T and STR*T instructions always perform register writeback and therefore never return a valid instruction syndrome.

When FEAT_RAS is implemented, ISV is 0 for any synchronous External abort.

ISV is set to 0 on a stage 2 abort on a stage 1 translation table walk.

When FEAT_RAS is not implemented, it is IMPLEMENTATION DEFINED whether ISV is set to 1 or 0 on a synchronous External abort on a stage 2 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SAS, bits [23:22]

Syndrome Access Size. When ISV is 1, indicates the size of the access attempted by the faulting operation.

SAS	Meaning
0b00	Byte
0b01	Halfword
0b10	Word
0b11	Doubleword

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SSE, bit [21]

Syndrome Sign Extend. When ISV is 1, for a byte, halfword, or word load operation, indicates whether the data item must be sign extended. For these cases, the possible values of this bit are:

SSE	Meaning
0b0	Sign-extension not required.
0b1	Data item must be sign-extended.

For all other operations this bit is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [20]

Reserved, RES0.

SRT, bits [19:16]

Syndrome Register Transfer. When ISV is 1, the register number of the Rt operand of the faulting instruction.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

AR, bit [14]

Acquire/Release. When ISV is 1, the possible values of this bit are:

AR	Meaning
0b0	Instruction did not have acquire/release semantics.
0b1	Instruction did have acquire/release semantics.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [13:12]

Reserved, RES0.

Bits[11:10]
When FEAT_RAS is implemented:

AET, bits [1:0] of bits [11:10]

Asynchronous Error Type. When DFSC is `0b010001`, describes the PE error state after taking the SError exception.

AET	Meaning
`0b00`	Uncontainable (UC).
`0b01`	Unrecoverable state (UEU).
`0b10`	Restartable state (UEO).
`0b11`	Recoverable state (UER).

On a synchronous Data Abort exception, this field is RES0.

In the event of multiple errors taken as a single SError exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

When FEAT_RAS is not implemented, or when DFSC is not `0b010001`:

- Bit[11] is RES0.
- Bit[10] forms the FnV field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Bit [1] of bits [11:10]

Reserved, RES0.

FnV, bit [0] of bits [11:10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
`0b0`	HDFAR is valid.
`0b1`	HDFAR is not valid, and holds an UNKNOWN value.

When FEAT_RAS is not implemented, this field is valid only if DFSC is `0b010000`. It is RES0 for all other aborts.

When FEAT_RAS is implemented:

- If DFSC is `0b010000`, this field is valid.
- If DFSC is `0b010001`, this bit forms part of the AET field, becoming AET[0].
- This field is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External Abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CM, bit [8]

Cache Maintenance. For a synchronous fault, identifies fault that comes from a cache maintenance or address translation instruction. For synchronous faults, the possible values of this bit are:

CM	Meaning
0b0	Fault not generated by a cache maintenance or address translation instruction.
0b1	Fault generated by a cache maintenance or address translation instruction.

For an asynchronous Data Abort exception, this bit is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

S1PTW	Meaning
0b0	Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1	Fault on the stage 2 translation of an access for a stage 1 translation table walk.

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by a write instruction or a read instruction.

WnR	Meaning
0b0	Abort caused by a read instruction.
0b1	Abort caused by a write instruction.

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

On an asynchronous Data Abort exception:

- When FEAT_RAS is not implemented, this bit is UNKNOWN.
- When FEAT_RAS is implemented, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code. Possible values of this field are:

DFSC	Meaning	Applies when
0b000000	Address size fault in translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk.	
0b010001	Asynchronous SError exception.	
0b010101	Synchronous External abort on translation table walk, level 1.	
0b010110	Synchronous External abort on translation table walk, level 2.	
0b010111	Synchronous External abort on translation table walk, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011001	Asynchronous SError exception, from a parity or ECC error on memory access.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk, level 3.	When FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Debug exception.	
0b110000	TLB conflict abort.	
0b110100	IMPLEMENTATION DEFINED fault (Lockdown).	
0b110101	IMPLEMENTATION DEFINED fault (Unsupported Exclusive access).	

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information for the ISS encoding for Exception from a Data Abort

The following describe cases where Data Abort exceptions can be routed to Hyp mode, generating exceptions that are reported in the HSR with EC value 0b100100:

- 'Abort exceptions, when the value of HCR.TGE is 1'.
- 'Routing debug exceptions to EL2 using AArch32'.

The following describe cases that can cause a Data Abort exception that is taken to Hyp mode, and reported in the HSR with EC value of 0b100000 or 0b100100:

- 'Hyp mode control of Non-secure access permissions'.
- 'Memory fault reporting in Hyp mode'.

Accessing HSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HSR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HSR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HSTR, Hyp System Trap Register

The HSTR characteristics are:

Purpose

Controls trapping to Hyp mode of Non-secure accesses, at EL1 or lower, to System registers in the coproc == 0b1111 encoding space:

- By the CRn value used to access the register using MCR or MRC instruction.
- By the CRm value used to access the register using MCRR or MRRC instruction.

Configuration

AArch32 System register HSTR bits [31:0] are architecturally mapped to AArch64 System register HSTR_EL2[31:0].

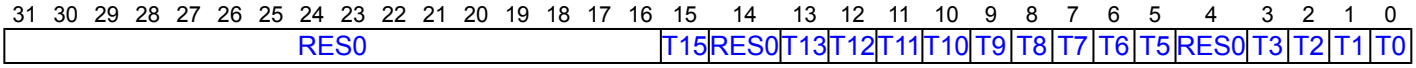
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HSTR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HSTR is a 32-bit register.

Field descriptions



Bits [31:16, 14, 4]

Reserved, RES0.

T<n>, bit [n], for n = 15, 13 to 5, 3 to 0

The remaining fields control whether Non-secure EL0 and EL1 accesses, using MCR or MRC instructions, reported using EC syndrome value 0x03, and MCRR or MRRC instructions, reported using EC syndrome value 0x04, to the System registers in the coproc == 0b1111 encoding space are trapped to Hyp mode:

T<n>	Meaning
0b0	This control has no effect on Non-secure EL0 or EL1 accesses to System registers.
0b1	Any Non-secure EL1 MCR or MRC access with coproc == 0b1111 and CRn == <n> is trapped to Hyp mode. A Non-secure EL0 MCR or MRC access with these values is trapped to Hyp mode only if the access is not UNDEFINED when the value of this field is 0. Any Non-secure EL1 MCRR or MRRC access with coproc == 0b1111 and CRm == <n> is trapped to Hyp mode. A Non-secure EL0 MCRR or MRRC access with these values is trapped to Hyp mode only if the access is not UNDEFINED when the value of this field is 0.

For example, when HSTR.T7 is 1, for instructions executed at Non-secure EL1:

- An MCR or MRC instruction with coproc set to 0b1111 and <CRn> set to c7 is trapped to Hyp mode.
- An MCRR or MRRC instruction with coproc set to 0b1111 and <CRm> set to c7 is trapped to Hyp mode.

The reset behavior of this field is:

- On a Warm reset:

- When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HSTR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HSTR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HSTR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0001	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HSTR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HSTR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HTCR, Hyp Translation Control Register

The HTCR characteristics are:

Purpose

The control register for stage 1 of the EL2 translation regime.

Note

This stage of translation always uses the Long-descriptor translation table format.

Configuration

AArch32 System register HTCR bits [31:0] are architecturally mapped to AArch64 System register TCR_EL2[31:0].

This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HTCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HTCR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22:14	13:12	11:10	9:8	7:6	5:3	2:0
RES1	IMPLEMENTATION DEFINED	RES0	HWU62	HWU61	HWU60	HWU59	HPD	RES1	RES0	SH0	ORGN0	IRGN0	RES0		T0SZ

Bit [31]

Reserved, RES1.

IMPLEMENTATION DEFINED, bit [30]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

HWU62	Meaning
0b0	Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1.

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

HWU61	Meaning
0b0	Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1.

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

HWU60	Meaning
0b0	Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1.

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1.

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD, bit [24]

When FEAT_AA32HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the PL2 translation regime.

HPD	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

Bits [22:14]

Reserved, RES0.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using HTTBR.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using HTTBR.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using HTTBR.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:3]

Reserved, RES0.

T0SZ, bits [2:0]

The size offset of the memory region addressed by HTTBR. The region size is $2^{(32-T0SZ)}$ bytes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HTCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HTCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HTCR;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HTCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HTCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HTPIDR, Hyp Software Thread ID Register

The HTPIDR characteristics are:

Purpose

Provides a location where software running in Hyp mode can store thread identifying information that is not visible to Non-secure software executing at EL0 or EL1, for hypervisor management purposes.

The PE makes no use of this register.

Configuration

AArch32 System register HTPIDR bits [31:0] are architecturally mapped to AArch64 System register TPIDR_EL2[31:0].

This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HTPIDR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

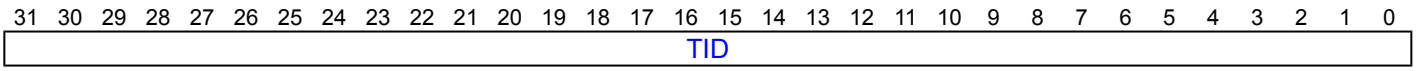
Note

The PE never updates this register.

Attributes

HTPIDR is a 32-bit register.

Field descriptions



TID, bits [31:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HTPIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HTPIDR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HTPIDR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HTPIDR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HTPIDR = R[t];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HTRFCR, Hyp Trace Filter Control Register

The HTRFCR characteristics are:

Purpose

Provides EL2 controls for Trace.

Configuration

AArch32 System register HTRFCR bits [31:0] are architecturally mapped to AArch64 System register TRFCR_EL2[31:0].

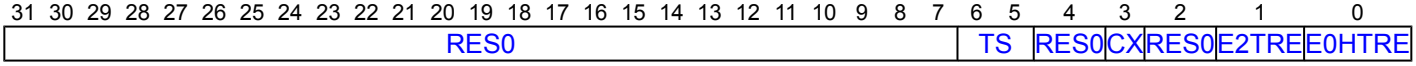
This register is present only when FEAT_AA32EL2 is implemented and FEAT_TRF is implemented. Otherwise, direct accesses to HTRFCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from Monitor mode when SCR.NS == 1.

Attributes

HTRFCR is a 32-bit register.

Field descriptions



Bits [31:7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control. Controls which timebase is used for trace timestamps.

TS	Meaning
0b00	The timestamp is controlled by TRFCR.TS.
0b01	Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF.
0b11	Physical timestamp. The traced timestamp is the physical counter value.

When SelfHostedTraceEnabled() == FALSE, this field is ignored.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [4]

Reserved, RES0.

CX, bit [3]

VMID Trace Enable.

CX	Meaning
0b0	VMID tracing is not allowed.
0b1	VMID tracing is allowed.

When SelfHostedTraceEnabled() == FALSE, this field is ignored.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E2TRE, bit [1]

EL2 Trace Enable.

E2TRE	Meaning
0b0	Tracing is prohibited at EL2.
0b1	Tracing is allowed at EL2.

When SelfHostedTraceEnabled() == FALSE, this field is ignored.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

E0HTRE, bit [0]

EL0 Trace Enable.

E0HTRE	Meaning
0b0	Tracing is prohibited at EL0 when HCR.TGE == 1.
0b1	Tracing is allowed at EL0 when HCR.TGE == 1.

This field is ignored if any of the following are true:

- The PE is in Secure state.
- SelfHostedTraceEnabled() == FALSE.
- HCR.TGE == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HTRFCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_TRF)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TTRF ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = HTRFCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HTRFCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_TRF)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TTRF ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        HTRFCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HTRFCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HTTBR, Hyp Translation Table Base Register

The HTTBR characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL2 translation regime, and other information for this translation regime.

Configuration

AArch32 System register HTTBR bits [47:0] are architecturally mapped to AArch64 System register TTBR0_EL2[47:0].

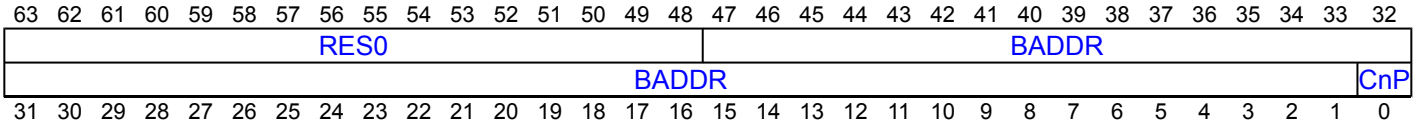
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HTTBR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HTTBR is a 64-bit register.

Field descriptions



Bits [63:48]

Reserved, RES0.

BADDR, bits [47:1]

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of HTCR.T0SZ as follows:

- If HTCR.T0SZ is 0 or 1, x = 5 - HTCR.T0SZ.
- If HTCR.T0SZ is greater than 1, x = 14 - HTCR.T0SZ.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by HTTBR is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of HTTBR.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by HTTBR are permitted to differ from corresponding entries for HTTBR for other PEs in the Inner Shareable domain. This is not affected by the value of HTTBR.CnP on those other PEs.
0b1	The translation table entries pointed to by HTTBR are the same as the translation table entries pointed to by HTTBR on every other PE in the Inner Shareable domain for which the value of HTTBR.CnP is 1.

Note

If the value of the HTTBR.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those HTTBRs do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing HTTBR

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b0010	0b0100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t, t2] = HTTBR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t, t2] = HTTBR;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b0010	0b0100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HTTBR = R[t, t2];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HTTBR = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HVBAR, Hyp Vector Base Address Register

The HVBAR characteristics are:

Purpose

Holds the vector base address for any exception that is taken to Hyp mode.

Configuration

AArch32 System register HVBAR bits [31:0] are architecturally mapped to AArch64 System register VBAR_EL2[31:0].

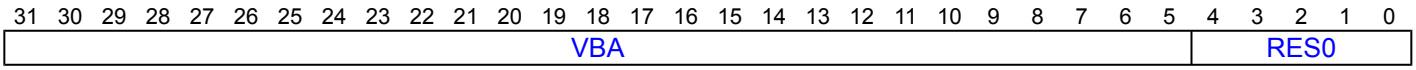
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to HVBAR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HVBAR is a 32-bit register.

Field descriptions



VBA, bits [31:5]

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:0]

Reserved, RES0.

Accessing HVBAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = HVBAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = HVBAR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HVBAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HVBAR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_AP0R<n>, Interrupt Controller Active Priorities Group 0 Registers, n = 0 - 3

The ICC_AP0R<n> characteristics are:

Purpose

Provides information about Group 0 active priorities.

Configuration

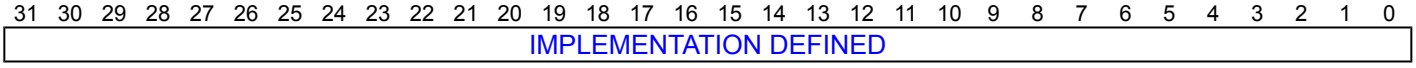
AArch32 System register ICC_AP0R<n> bits [31:0] are architecturally mapped to AArch64 System register ICC_AP0R<n>_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_AP0R<n> are UNDEFINED.

Attributes

ICC_AP0R<n> is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00000000000000000000000000000000'.

Additional information

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICC_AP0R<n>

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 0 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP0R1 is implemented only in implementations that support 6 or more bits of preemption. ICC_AP0R2 and ICC_AP0R3 are implemented only in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR.PREbits.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICC_AP0R<n>.
- Secure ICC_AP1R<n>.
- Non-secure ICC_AP1R<n>.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b1:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_AP0R[m];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_AP0R[m];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_AP0R[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_AP0R[m];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICC_AP0R[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b1:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_AP0R[m] = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_AP0R[m] = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_AP0R[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_AP0R[m] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_AP0R[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_AP1R<n>, Interrupt Controller Active Priorities Group 1 Registers, n = 0 - 3

The ICC_AP1R<n> characteristics are:

Purpose

Provides information about Group 1 active priorities.

Configuration

This register is banked between ICC_AP1R<n> and ICC_AP1R<n>_S and ICC_AP1R<n>_NS.

AArch32 System register ICC_AP1R<n> bits [31:0] (ICC_AP1R<n>_S) are architecturally mapped to AArch64 System register ICC_AP1R<n>_EL1[31:0] (ICC_AP1R<n>_EL1_S).

AArch32 System register ICC_AP1R<n> bits [31:0] (ICC_AP1R<n>_NS) are architecturally mapped to AArch64 System register ICC_AP1R<n>_EL1[31:0] (ICC_AP1R<n>_EL1_NS).

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_AP1R<n> are UNDEFINED.

Attributes

ICC_AP1R<n> is a 32-bit register.

This register has the following instances:

- ICC_AP1R<n>, when EL3 is not implemented or FEAT_AA64 is implemented.
- ICC_AP1R<n>_S, when FEAT_AA32EL3 is implemented.
- ICC_AP1R<n>_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMPLEMENTATION DEFINED																															

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00000000000000000000000000000000'.

Additional information

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICC_AP1R<n>

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP1R1 is implemented only in implementations that support 6 or more bits of preemption. ICC_AP1R2 and ICC_AP1R3 are implemented only in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR.PREbits.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICC_AP0R<n>
- Secure ICC_AP1R<n>
- Non-secure ICC_AP1R<n>

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_AP1R[m];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_AP1R[m];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_AP1R_NS[m];
    else
        R[t] = ICC_AP1R[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_AP1R_NS[m];
    else
        R[t] = ICC_AP1R[m];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            R[t] = ICC_AP1R_S[m];
        else
            R[t] = ICC_AP1R_NS[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_AP1R[m] = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_AP1R[m] = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_AP1R_NS[m] = R[t];
    else
        ICC_AP1R[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_AP1R_NS[m] = R[t];
    else
        ICC_AP1R[m] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            ICC_AP1R_S[m] = R[t];
        else
            ICC_AP1R_NS[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_ASGI1R, Interrupt Controller Alias Software Generated Interrupt Group 1 Register

The ICC_ASGI1R characteristics are:

Purpose

Generates Group 1 SGIs for the Security state that is not the current Security state.

Configuration

AArch32 System register ICC_ASGI1R performs the same function as AArch64 System register ICC_ASGI1R_EL1.

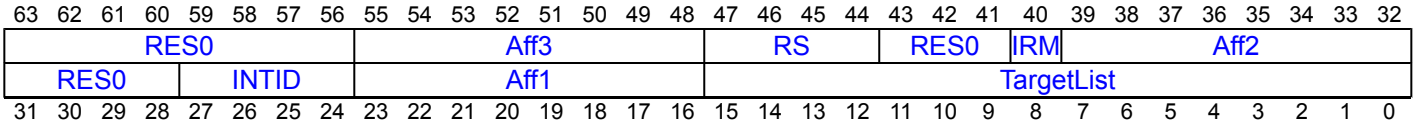
This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_ASGI1R are UNDEFINED.

Under certain conditions a write to ICC_ASGI1R can generate Group 0 interrupts, see 'Forwarding an SGI to a target PE' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_ASGI1R is a 64-bit register.

Field descriptions



Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value ((RS * 16) + n).

When ICC_CTLR_EL1.RSS==0, RS is RES0.

When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a CONSTRAINED UNPREDICTABLE choice of:

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

IRM	Meaning
0b0	Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>.
0b1	Interrupts routed to all PEs in the system, excluding "self".

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

Note

If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Accessing ICC_ASGI1R

This register allows software executing in a Secure state to generate Non-secure Group 1 SGIs. It will also allow software executing in a Non-secure state to generate Secure Group 1 SGIs, if permitted by the settings of GICR_NSACR in the Redistributor corresponding to the target PE.

When GICD_CTLR.DS==0, Non-secure writes do not generate an interrupt for a target PE if not permitted by the GICR_NSACR register associated with the target PE. For more information, see 'Use of control registers for SGI forwarding' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Accesses from Secure Monitor mode are treated as Secure regardless of the value of SCR.NS.

Accesses to this register use the following encodings in the System register encoding space:

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1100	0b0001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_ASGI1R = R[t, t2];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_ASGI1R = R[t, t2];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_ASGI1R = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_BPR0, Interrupt Controller Binary Point Register 0

The ICC_BPR0 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

Configuration

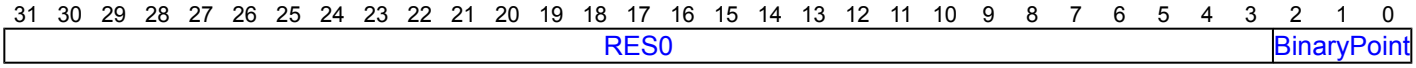
AArch32 System register ICC_BPR0 bits [31:0] are architecturally mapped to AArch64 System register ICC_BPR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_BPR0 are UNDEFINED.

Attributes

ICC_BPR0 is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

Binary point value	Group priority field	Subpriority field	Field with binary point
0	[7:1]	[0]	ggggggg.s
1	[7:2]	[1:0]	gggggg.ss
2	[7:3]	[2:0]	ggggg.sss
3	[7:4]	[3:0]	gggg.ssss
4	[7:5]	[4:0]	ggg.sssss
5	[7:6]	[5:0]	gg.ssssss
6	[7]	[6:0]	g.sssssss
7	No preemption	[7:0]	.sssssss

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_BPR0

The minimum binary point value is derived from the number of implemented priority bits. The number of priority bits is IMPLEMENTATION DEFINED, and reported by ICC_CTLR.PRIbits and ICC_MCTLR.PRIbits.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is set to the minimum supported value.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_BPR0;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_BPR0;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_BPR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_BPR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_BPR0;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_BPR0 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_BPR0 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_BPR0 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_BPR0 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        ICC_BPR0 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_BPR1, Interrupt Controller Binary Point Register 1

The ICC_BPR1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

Configuration

This register is banked between ICC_BPR1 and ICC_BPR1_S and ICC_BPR1_NS.

AArch32 System register ICC_BPR1 bits [31:0] (ICC_BPR1_S) are architecturally mapped to AArch64 System register ICC_BPR1_EL1[31:0] (ICC_BPR1_EL1_S).

AArch32 System register ICC_BPR1 bits [31:0] (ICC_BPR1_NS) are architecturally mapped to AArch64 System register ICC_BPR1_EL1[31:0] (ICC_BPR1_EL1_NS).

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_BPR1 are UNDEFINED.

In GIC implementations supporting two Security states, this register is Banked.

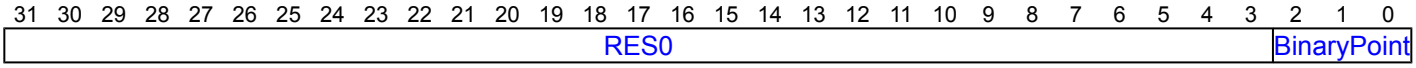
Attributes

ICC_BPR1 is a 32-bit register.

This register has the following instances:

- ICC_BPR1, when EL3 is not implemented or FEAT_AA64 is implemented.
- ICC_BPR1_S, when FEAT_AA32EL3 is implemented.
- ICC_BPR1_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



Bits [31:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

If the GIC is configured to use separate binary point fields for Group 0 and Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. For more information about priorities, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Writing 0 to this field will set this field to its reset value.

If EL3 is implemented and ICC_MCTLR.CBPR_EL1S is 1:

- Accesses to this register at EL3 not in Monitor mode access the state of ICC_BPR0.
- When SCR_EL3.EEL2 is 1 and HCR_EL2.IMO is 1, Secure accesses to this register at EL1 access the state of ICV_BPR1.
- Otherwise, Secure accesses to this register at EL1 access the state of ICC_BPR0.

If EL3 is implemented and ICC_MCTLR.CBPR_EL1NS is 1, Non-secure accesses to this register at EL1 and EL2 behave as follows, depending on the values of HCR.IMO and SCR.IRQ:

HCR.IMO	SCR_IRQ	Behavior
0b0	0b0	Non-secure EL1 and EL2 reads return ICC_BPR0 + 1 saturated to 0b111. Non-secure EL1 and EL2 writes are ignored.
0b0	0b1	Non-secure EL1 and EL2 accesses trap to EL3.
0b1	0b0	Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 reads return ICC_BPR0 + 1 saturated to 0b111. Non-secure EL2 writes ignored.
0b1	0b1	Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 accesses trap to EL3.

If EL3 is not implemented and ICC_CTLR.CBPR is 1, Non-secure accesses to this register at EL1 and EL2 behave as follows, depending on the values of HCR.IMO:

HCR.IMO	Behavior
0b0	Non-secure EL1 and EL2 reads return ICC_BPR0 + 1 saturated to 0b111. Non-secure EL1 and EL2 writes are ignored.
0b1	Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 reads return ICC_BPR0 + 1 saturated to 0b111. Non-secure EL2 writes are ignored.

This field resets to an IMPLEMENTATION DEFINED nonzero value.

Accessing ICC_BPR1

When the PE resets into an Exception level that is using AArch32, the reset value is equal to:

- For the Secure copy of the register, the minimum value of ICC_BPR0 plus one.
- For the Non-secure copy of the register, the minimum value of ICC_BPR0.

Where the minimum value of ICC_BPR0 is IMPLEMENTATION DEFINED.

If EL3 is not implemented:

- If the PE is Secure this reset value is (minimum value of ICC_BPR0 plus one).
- If the PE is Non-secure this reset value is (minimum value of ICC_BPR0).

An attempt to program the binary point field to a value less than the reset value sets the field to the reset value.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_BPR1;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_BPR1;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_BPR1_NS;
    else
        R[t] = ICC_BPR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_BPR1_NS;
```

```
    else
        R[t] = ICC_BPR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            R[t] = ICC_BPR1_S;
        else
            R[t] = ICC_BPR1_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_BPR1 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_BPR1 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_BPR1_NS = R[t];
    else
        ICC_BPR1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_BPR1_NS = R[t];
```

```
    else
        ICC_BPR1 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            ICC_BPR1_S = R[t];
        else
            ICC_BPR1_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_CTLR, Interrupt Controller Control Register

The ICC_CTLR characteristics are:

Purpose

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Configuration

This register is banked between ICC_CTLR and ICC_CTLR_S and ICC_CTLR_NS.

AArch32 System register ICC_CTLR bits [31:0] (ICC_CTLR_S) are architecturally mapped to AArch64 System register ICC_CTLR_EL1[31:0] (ICC_CTLR_EL1_S).

AArch32 System register ICC_CTLR bits [31:0] (ICC_CTLR_NS) are architecturally mapped to AArch64 System register ICC_CTLR_EL1[31:0] (ICC_CTLR_EL1_NS).

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_CTLR are UNDEFINED.

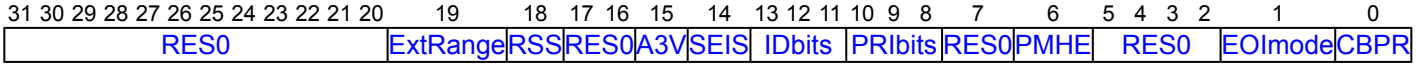
Attributes

ICC_CTLR is a 32-bit register.

This register has the following instances:

- ICC_CTLR, when EL3 is not implemented or FEAT_AA64 is implemented.
- ICC_CTLR_S, when FEAT_AA32EL3 is implemented.
- ICC_CTLR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



Bits [31:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ExtRange	Meaning
0b0	CPU interface does not support INTIDs in the range 1024..8191. Behavior is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface. **Note** Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.
0b1	CPU interface supports INTIDs in the range 1024..8191. All INTIDs in the range 1024..8191 are treated as requiring deactivation.

If EL3 is implemented, ICC_CTLR_EL1.ExtRange is an alias of ICC_CTLR_EL3.ExtRange.

Access to this field is **RO**.

RSS, bit [18]

Range Selector Support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RSS	Meaning
0b0	Targeted SGIs with affinity level 0 values of 0 - 15 are supported.
0b1	Targeted SGIs with affinity level 0 values of 0 - 255 are supported.

Access to this field is **RO**.

Bits [17:16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

A3V	Meaning
0b0	The CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.
0b1	The CPU interface logic supports nonzero values of Affinity 3 in SGI generation System registers.

If EL3 is implemented and using AArch32, this bit is an alias of ICC_MCTLR.A3V.

If EL3 is implemented and using AArch64, this bit is an alias of ICC_CTLR_EL3.A3V.

Access to this field is **RO**.

SEIS, bit [14]

SEI Support. Indicates whether the CPU interface supports local generation of SEIs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEIS	Meaning
0b0	The CPU interface logic does not support local generation of SEIs.
0b1	The CPU interface logic supports local generation of SEIs.

If EL3 is implemented and using AArch32, this bit is an alias of ICC_MCTLR.SEIS.

If EL3 is implemented and using AArch64, this bit is an alias of ICC_CTLR_EL3.SEIS.

Access to this field is **RO**.

IDbits, bits [13:11]

Identifier bits. The number of physical interrupt identifier bits supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IDbits	Meaning
0b000	16 bits.
0b001	24 bits.

All other values are reserved.

If EL3 is implemented and using AArch32, this field is an alias of ICC_MCTLR.IDbits.

If EL3 is implemented and using AArch64, this field is an alias of ICC_CTLR_EL3.IDbits.

Access to this field is **RO**.

PRIbits, bits [10:8]

Priority bits. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

Note

This field always returns the number of priority bits implemented, regardless of the Security state of the access or the value of GICD_CTLR.DS.

The division between group priority and subpriority is defined in the binary point registers ICC_BPR0 and ICC_BPR1.

If EL3 is implemented and using AArch32, physical accesses return the value from ICC_MCTLR.PRIbits.

If EL3 is implemented and using AArch64, physical accesses return the value from ICC_CTLR_EL3.PRIbits.

If EL3 is not implemented, physical accesses return the value from this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [7]

Reserved, RES0.

PMHE, bit [6]

Priority Mask Hint Enable. Controls whether the priority mask register is used as a hint for interrupt distribution:

PMHE	Meaning
0b0	Disables use of ICC_PMR as a hint for interrupt distribution.
0b1	Enables use of ICC_PMR as a hint for interrupt distribution.

If EL3 is implemented:

- If EL3 is using AArch32, this bit is an alias of ICC_MCTLR.PMHE.
- If EL3 is using AArch64, this bit is an alias of ICC_CTLR_EL3.PMHE.
- If GICD_CTLR.DS == 0, this bit is read-only.
- If GICD_CTLR.DS == 1, this bit is read/write.

If EL3 is not implemented, it is IMPLEMENTATION DEFINED whether this bit is read-only or read/write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

Bits [5:2]

Reserved, RES0.

EOImode, bit [1]

EOI mode for the current Security state. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

EOImode	Meaning
0b0	ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE.
0b1	ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality.

If EL3 is implemented:

- If EL3 is using AArch32, this bit is an alias of ICC_MCTLR.EOImode_EL1{S, NS} where S or NS corresponds to the current Security state.
- If EL3 is using AArch64, this bit is an alias of ICC_CTLR_EL3.EOImode_EL1{S, NS} where S or NS corresponds to the current Security state.

If EL3 is not implemented, it is IMPLEMENTATION DEFINED whether this bit is read-only or read/write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

CBPR, bit [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 interrupts:

CBPR	Meaning
0b0	ICC_BPR0 determines the preemption group for Group 0 interrupts only. ICC_BPR1 determines the preemption group for Group 1 interrupts.
0b1	ICC_BPR0 determines the preemption group for both Group 0 and Group 1 interrupts.

If EL3 is implemented:

- If EL3 is using AArch32, this bit is an alias of ICC_MCTLR.CBPR_EL1{S,NS} where S or NS corresponds to the current Security state.
- If EL3 is using AArch64, this bit is an alias of ICC_CTLR_EL3.CBPR_EL1{S,NS} where S or NS corresponds to the current Security state.
- If GICD_CTLR.DS == 0, this bit is read-only.
- If GICD_CTLR.DS == 1, this bit is read/write.

If EL3 is not implemented, it is IMPLEMENTATION DEFINED whether this bit is read-only or read/write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

Accessing ICC_CTLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_CTLR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_CTLR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_CTLR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_CTLR;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_CTLR_NS;
    else
        R[t] = ICC_CTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_CTLR_NS;
    else
        R[t] = ICC_CTLR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            R[t] = ICC_CTLR_S;
        else
            R[t] = ICC_CTLR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_CTLR = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_CTLR_NS = R[t];
    else
        ICC_CTLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_CTLR_NS = R[t];
    else
        ICC_CTLR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            ICC_CTLR_S = R[t];
        else
            ICC_CTLR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_DIR, Interrupt Controller Deactivate Interrupt Register

The ICC_DIR characteristics are:

Purpose

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified interrupt.

Configuration

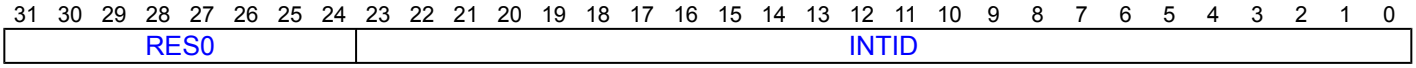
AArch32 System register ICC_DIR bits [31:0] performs the same function as AArch64 System register ICC_DIR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_DIR are UNDEFINED.

Attributes

ICC_DIR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_DIR

There are two cases when writing to ICC_DIR_EL1 that were UNPREDICTABLE for a corresponding GICv2 write to GICC_DIR:

- When EOImode == 0. GICv3 implementations must ignore such writes. In systems supporting system error generation, an implementation might generate an SEI.
- When EOImode == 1 but no EOI has been issued. The interrupt will be deactivated by the Distributor, however the active priority in the CPU interface for the interrupt will remain set (because no EOI was issued).

Accesses to this register use the following encodings in the System register encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1011	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TDIR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TDIR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_DIR = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_DIR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_DIR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_DIR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_EOIR0, Interrupt Controller End Of Interrupt Register 0

The ICC_EOIR0 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 0 interrupt.

Configuration

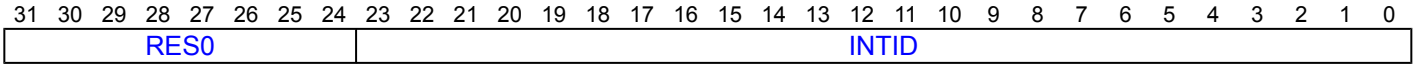
AArch32 System register ICC_EOIR0 bits [31:0] performs the same function as AArch64 System register ICC_EOIR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_EOIR0 are UNDEFINED.

Attributes

ICC_EOIR0 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding ICC_IAR0 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to ICC_DIR to deactivate the interrupt.

The appropriate EOImode bit varies as follows:

- If EL3 is not implemented, the appropriate bit is ICC_CTLR.EOImode.
- If EL3 is implemented and the software is executing in Monitor mode, the appropriate bit is ICC_MCTLR.EOImode_EL3.
- If EL3 is implemented and the software is not executing in Monitor mode, the bit depends on the current Security state:
    - If the software is executing in Secure state, the bit is ICC_CTLR.EOImode in the Secure instance of ICC_CTLR. This is an alias of ICC_MCTLR.EOImode_EL1S.
    - If the software is executing in Non-secure state, the bit is ICC_CTLR.EOImode in the Non-secure instance of ICC_CTLR. This is an alias of ICC_MCTLR.EOImode_EL1NS.

Accessing ICC_EOIR0

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICC_IAR0, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accesses to this register use the following encodings in the System register encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_EOIR0 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_EOIR0 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR0 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR0 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        ICC_EOIR0 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_EOIR1, Interrupt Controller End Of Interrupt Register 1

The ICC_EOIR1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 1 interrupt.

Configuration

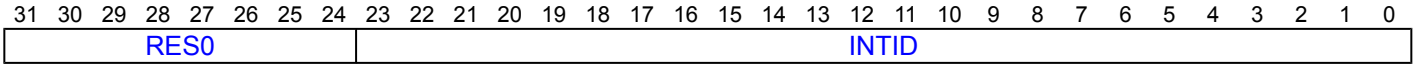
AArch32 System register ICC_EOIR1 bits [31:0] performs the same function as AArch64 System register ICC_EOIR1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_EOIR1 are UNDEFINED.

Attributes

ICC_EOIR1 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding ICC_IAR1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to ICC_DIR to deactivate the interrupt.

The appropriate EOImode bit varies as follows:

- If EL3 is not implemented, the appropriate bit is ICC_CTLR.EOImode.
- If EL3 is implemented and the software is executing in Monitor mode, the appropriate bit is ICC_MCTLR.EOImode_EL3.
- If EL3 is implemented and the software is not executing in Monitor mode, the bit depends on the current Security state:
    - If the software is executing in Secure state, the bit is ICC_CTLR.EOImode in the Secure instance of ICC_CTLR. This is an alias of ICC_MCTLR.EOImode_EL1S.
    - If the software is executing in Non-secure state, the bit is ICC_CTLR.EOImode in the Non-secure instance of ICC_CTLR. This is an alias of ICC_MCTLR.EOImode_EL1NS.

Accessing ICC_EOIR1

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICC_IAR1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accesses to this register use the following encodings in the System register encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_EOIR1 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_EOIR1 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        ICC_EOIR1 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_HPPIR0, Interrupt Controller Highest Priority Pending Interrupt Register 0

The ICC_HPPIR0 characteristics are:

Purpose

Indicates the highest priority pending Group 0 interrupt on the CPU interface.

Configuration

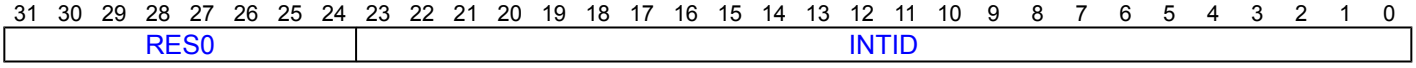
AArch32 System register ICC_HPPIR0 bits [31:0] performs the same function as AArch64 System register ICC_HPPIR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_HPPIR0 are UNDEFINED.

Attributes

ICC_HPPIR0 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_HPPIR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b010

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_HPPIR0;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_HPPIR0;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_HPPIR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_HPPIR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_HPPIR0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_HPPIR1, Interrupt Controller Highest Priority Pending Interrupt Register 1

The ICC_HPPIR1 characteristics are:

Purpose

Indicates the highest priority pending Group 1 interrupt on the CPU interface.

Configuration

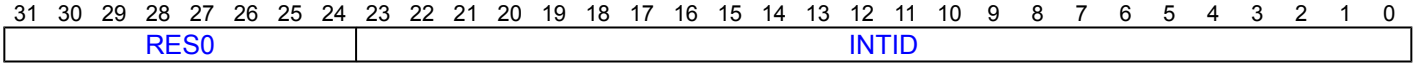
AArch32 System register ICC_HPPIR1 bits [31:0] performs the same function as AArch64 System register ICC_HPPIR1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_HPPIR1 are UNDEFINED.

Attributes

ICC_HPPIR1 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_HPPIR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_HPPIR1;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_HPPIR1;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_HPPIR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_HPPIR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_HPPIR1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_HSRE, Interrupt Controller Hyp System Register Enable register

The ICC_HSRE characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL2.

Configuration

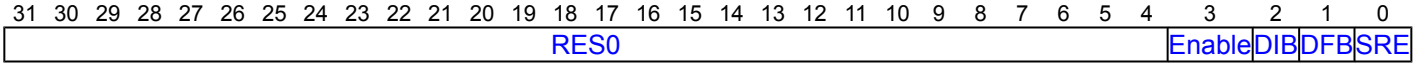
AArch32 System register ICC_HSRE bits [31:0] are architecturally mapped to AArch64 System register ICC_SRE_EL2[31:0].

This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICC_HSRE are UNDEFINED.

Attributes

ICC_HSRE is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

Enable, bit [3]

Enable. Enables lower Exception level access to ICC_SRE.

Enable	Meaning
0b0	Non-secure EL1 accesses to ICC_SRE trap to EL2.
0b1	Non-secure EL1 accesses to ICC_SRE do not trap to EL2.

If ICC_HSRE.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_HSRE.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIB, bit [2]

Disable IRQ bypass.

DIB	Meaning
0b0	IRQ bypass enabled.
0b1	IRQ bypass disabled.

If EL3 is implemented and GICD_CTLR.DS is 0, this field is a read-only alias of ICC_MSRE.DIB.

If EL3 is implemented and GICD_CTLR.DS is 1, this field is a read/write alias of ICC_MSRE.DIB.

In systems that do not support IRQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

DFB, bit [1]

Disable FIQ bypass.

DFB	Meaning
0b0	FIQ bypass enabled.
0b1	FIQ bypass disabled.

If EL3 is implemented and GICD_CTLR.DS is 0, this field is a read-only alias of ICC_MSRE.DFB.

If EL3 is implemented and GICD_CTLR.DS is 1, this field is a read/write alias of ICC_MSRE.DFB.

In systems that do not support FIQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

SRE, bit [0]

System Register Enable.

SRE	Meaning
0b0	The memory-mapped interface must be used. Accesses at EL2 or below to any ICH_* System register, or any EL1 or EL2 ICC_* register other than ICC_SRE or ICC_HSRE, are UNDEFINED.
0b1	The System register interface to the ICH_* registers and the EL1 and EL2 ICC_* registers is enabled for EL2.

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and using AArch64:

- When ICC_SRE_EL3.SRE==0 this bit is RAZ/WI.

If EL3 is implemented using AArch32:

- When ICC_MSRE.SRE==0 this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICC_HSRE

The GIC architecture permits, but does not require, that registers can be shared between memory-mapped registers and the equivalent System registers. This means that if the memory-mapped registers have been accessed while ICC_HSRE.SRE==0, then the System registers might be modified. Therefore, software must only rely on the reset values of the System registers if there has been no use of the GIC functionality while the memory-mapped registers are in use. Otherwise, the System register values must be treated as UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1001	0b101

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
ICC_MSRE.Enable == '0' then
        UNDEFINED;
    else
        R[t] = ICC_HSRE;
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        R[t] = ICC_HSRE;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1001	0b101

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
ICC_MSRE.Enable == '0' then
        UNDEFINED;
    else
        ICC_HSRE = R[t];
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        ICC_HSRE = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IAR0, Interrupt Controller Interrupt Acknowledge Register 0

The ICC_IAR0 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled Group 0 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch32 System register ICC_IAR0 bits [31:0] performs the same function as AArch64 System register ICC_IAR0_EL1[31:0].

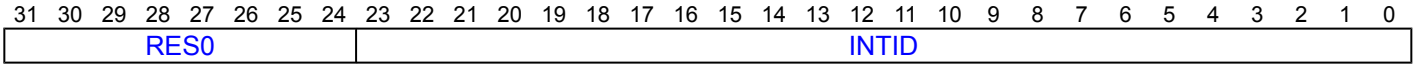
This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_IAR0 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronizing when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_IAR0 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_IAR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b1100	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_IAR0;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_IAR0;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IAR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IAR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_IAR0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IAR1, Interrupt Controller Interrupt Acknowledge Register 1

The ICC_IAR1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch32 System register ICC_IAR1 bits [31:0] performs the same function as AArch64 System register ICC_IAR1_EL1[31:0].

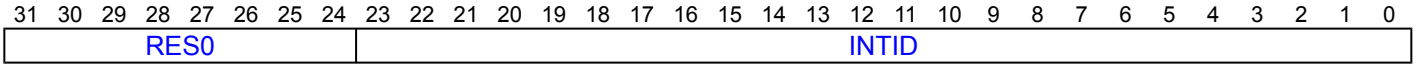
This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_IAR1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronizing when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_IAR1 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_IAR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b1100	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_IAR1;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_IAR1;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IAR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IAR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_IAR1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IGRPEN0, Interrupt Controller Interrupt Group 0 Enable register

The ICC_IGRPEN0 characteristics are:

Purpose

Controls whether Group 0 interrupts are enabled or not.

Configuration

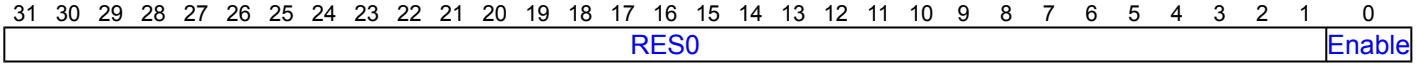
AArch32 System register ICC_IGRPEN0 bits [31:0] are architecturally mapped to AArch64 System register ICC_IGRPEN0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_IGRPEN0 are UNDEFINED.

Attributes

ICC_IGRPEN0 is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

Enable, bit [0]

Enables Group 0 interrupts.

Enable	Meaning
0b0	Group 0 interrupts are disabled.
0b1	Group 0 interrupts are enabled.

Virtual accesses to this register update ICH_VMCR.VENG0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICC_IGRPEN0

The lowest Exception level at which this register can be accessed is governed by the Exception level to which FIQ is routed. This routing depends on SCR.FIQ, SCR.NS and HCR.FMO.

If an interrupt is pending within the CPU interface when Enable becomes 0, the interrupt must be released to allow the Distributor to forward the interrupt to a different PE.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_IGRPEN0;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_IGRPEN0;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IGRPEN0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IGRPEN0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_IGRPEN0;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_IGRPEN0 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_IGRPEN0 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_IGRPEN0 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_IGRPEN0 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        ICC_IGRPEN0 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IGRPEN1, Interrupt Controller Interrupt Group 1 Enable register

The ICC_IGRPEN1 characteristics are:

Purpose

Controls whether Group 1 interrupts are enabled for the current Security state.

Configuration

This register is banked between ICC_IGRPEN1 and ICC_IGRPEN1_S and ICC_IGRPEN1_NS.

AArch32 System register ICC_IGRPEN1 bits [31:0] (ICC_IGRPEN1_S) are architecturally mapped to AArch64 System register ICC_IGRPEN1_EL1[31:0] (ICC_IGRPEN1_EL1_S).

AArch32 System register ICC_IGRPEN1 bits [31:0] (ICC_IGRPEN1_NS) are architecturally mapped to AArch64 System register ICC_IGRPEN1_EL1[31:0] (ICC_IGRPEN1_EL1_NS).

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_IGRPEN1 are UNDEFINED.

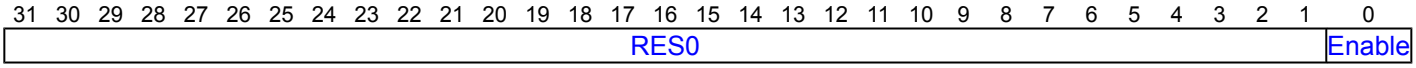
Attributes

ICC_IGRPEN1 is a 32-bit register.

This register has the following instances:

- ICC_IGRPEN1, when EL3 is not implemented or FEAT_AA64 is implemented.
- ICC_IGRPEN1_S, when FEAT_AA32EL3 is implemented.
- ICC_IGRPEN1_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



Bits [31:1]

Reserved, RES0.

Enable, bit [0]

Enables Group 1 interrupts for the current Security state.

Enable	Meaning
0b0	Group 1 interrupts are disabled for the current Security state.
0b1	Group 1 interrupts are enabled for the current Security state.

Virtual accesses to this register update ICH_VMCR.VENG1.

If EL3 is present:

- This bit is a read/write alias of ICC_MGRPEN1.EnableGrp1{S, NS} as appropriate if EL3 is using AArch32, or ICC_IGRPEN1_EL3.EnableGrp1{S, NS} as appropriate if EL3 is using AArch64.
- When this register is accessed at EL3, the copy of this register appropriate to the current setting of SCR.NS is accessed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICC_IGRPEN1

The lowest Exception level at which this register can be accessed is governed by the Exception level to which IRQ is routed. This routing depends on the IRQ and NS fields of SCR or SCR_EL3 and the IMO field of HCR or HCR_EL2.

If an interrupt is pending within the CPU interface when Enable becomes 0, the interrupt must be released to allow the Distributor to forward the interrupt to a different PE.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_IGRPEN1;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_IGRPEN1;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_IGRPEN1_NS;
    else
        R[t] = ICC_IGRPEN1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_IGRPEN1_NS;
```

```
    else
        R[t] = ICC_IGRPEN1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            R[t] = ICC_IGRPEN1_S;
        else
            R[t] = ICC_IGRPEN1_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_IGRPEN1 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_IGRPEN1 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_IGRPEN1_NS = R[t];
    else
        ICC_IGRPEN1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_IGRPEN1_NS = R[t];
```

```
    else
        ICC_IGRPEN1 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            ICC_IGRPEN1_S = R[t];
        else
            ICC_IGRPEN1_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_MCTLR, Interrupt Controller Monitor Control Register

The ICC_MCTLR characteristics are:

Purpose

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

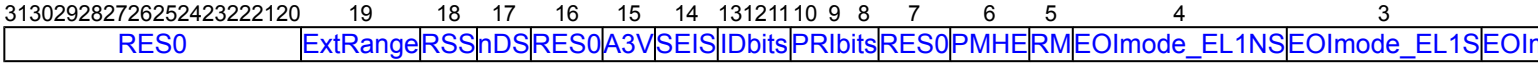
Configuration

This register is present only when FEAT_AA32EL3 is implemented, GICv3 is implemented, and EL3 is implemented. Otherwise, direct accesses to ICC_MCTLR are UNDEFINED.

Attributes

ICC_MCTLR is a 32-bit register.

Field descriptions



Bits [31:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

ExtRange	Meaning
0b0	CPU interface does not support INTIDs in the range 1024..8191. Behavior is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface. **Note** Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.
0b1	CPU interface supports INTIDs in the range 1024..8191 All INTIDs in the range 1024..8191 are treated as requiring deactivation.

RSS, bit [18]

Range Selector Support. Possible values are:

RSS	Meaning
0b0	Targeted SGIs with affinity level 0 values of 0 - 15 are supported.
0b1	Targeted SGIs with affinity level 0 values of 0 - 255 are supported.

This bit is read-only.

nDS, bit [17]

Disable Security not supported. Read-only and writes are ignored.

nDS	Meaning
0b0	The CPU interface logic supports disabling of security.
0b1	The CPU interface logic does not support disabling of security, and requires that security is not disabled.

Bit [16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored.

A3V	Meaning
0b0	The CPU interface logic does not support nonzero values of the Aff3 field in SGI generation System registers.
0b1	The CPU interface logic supports nonzero values of the Aff3 field in SGI generation System registers.

If EL3 is present, ICC_CTLR.A3V is an alias of ICC_MCTLR.A3V

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports generation of SEIs.

SEIS	Meaning
0b0	The CPU interface logic does not support generation of SEIs.
0b1	The CPU interface logic supports generation of SEIs.

If EL3 is present, ICC_CTLR.SEIS is an alias of ICC_MCTLR.SEIS

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. Indicates the number of physical interrupt identifier bits supported.

IDbits	Meaning
0b000	16 bits.
0b001	24 bits.

All other values are reserved.

If EL3 is present, ICC_CTLR.IDbits is an alias of ICC_MCTLR.IDbits

PRIbits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

Note

This field always returns the number of priority bits implemented, regardless of the value of SCR.NS or the value of GICD_CTLR.DS.

The division between group priority and subpriority is defined in the binary point registers ICC_BPR0 and ICC_BPR1.

This field determines the minimum value of ICC_BPR0.

Bit [7]

Reserved, RES0.

PMHE, bit [6]

Priority Mask Hint Enable.

PMHE	Meaning
0b0	Disables use of the priority mask register as a hint for interrupt distribution.
0b1	Enables use of the priority mask register as a hint for interrupt distribution.

Software must write ICC_PMR to 0xFF before clearing this field to 0.

An implementation might choose to make this field RAO/WI.

If EL3 is present, ICC_CTLR.PMHE is an alias of ICC_MCTLR.PMHE.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

RM, bit [5]

SBZ.

The equivalent bit in AArch64 is the Routing Modifier bit. This feature is not supported when EL3 is using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL1NS, bit [4]

EOI mode for interrupts handled at Non-secure EL1 and EL2. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOImode_EL1NS	Meaning
0b0	ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE.
0b1	ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality.

If EL3 is present, ICC_CTLR(NS).EOImode is an alias of ICC_MCTLR.EOImode_EL1NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL1S, bit [3]

EOI mode for interrupts handled at Secure EL1. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOImode_EL1S	Meaning
0b0	ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE.
0b1	ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality.

If EL3 is present, ICC_CTLR(S).EOImode is an alias of ICC_MCTLR.EOImode_EL1S.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL3, bit [2]

EOI mode for interrupts handled at EL3. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOImode_EL3	Meaning
0b0	ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE.
0b1	ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR_EL1NS, bit [1]

Common Binary Point Register, EL1 Non-secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Non-secure interrupts at EL1 and EL2.

CBPR_EL1NS	Meaning
0b0	ICC_BPR0 determines the preemption group for Group 0 interrupts only. ICC_BPR1 determines the preemption group for Non-secure Group 1 interrupts.
0b1	ICC_BPR0 determines the preemption group for Group 0 interrupts and Non-secure Group 1 interrupts. Non-secure accesses to GICC_BPR and ICC_BPR1 access the state of ICC_BPR0.

If EL3 is present, ICC_CTLR(NS).CBPR is an alias of ICC_MCTLR.CBPR_EL1NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR_EL1S, bit [0]

Common Binary Point Register, EL1 Secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Secure interrupts in Secure non-Monitor modes.

CBPR_EL1S	Meaning
0b0	ICC_BPR0 determines the preemption group for Group 0 interrupts only. ICC_BPR1 determines the preemption group for Secure Group 1 interrupts.
0b1	ICC_BPR0 determines the preemption group for Group 0 interrupts and Secure Group 1 interrupts. Secure EL1 accesses, or EL3 accesses when not in Monitor mode, to ICC_BPR1 access the state of ICC_BPR0.

If EL3 is present, ICC_CTLR(S).CBPR is an alias of ICC_MCTLR.CBPR_EL1S.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_MCTLR

This register is only accessible when executing in Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b110	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_AA32EL3) && IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICC_MCTLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b110	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_AA32EL3) && IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_MCTLR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_MGRPEN1, Interrupt Controller Monitor Interrupt Group 1 Enable register

The ICC_MGRPEN1 characteristics are:

Purpose

Controls whether Group 1 interrupts are enabled or not.

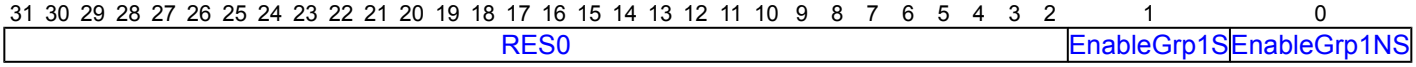
Configuration

This register is present only when FEAT_AA32EL3 is implemented, GICv3 is implemented, and EL3 is implemented. Otherwise, direct accesses to ICC_MGRPEN1 are UNDEFINED.

Attributes

ICC_MGRPEN1 is a 32-bit register.

Field descriptions



Bits [31:2]

Reserved, RES0.

EnableGrp1S, bit [1]

Enables Group 1 interrupts for the Secure state.

EnableGrp1S	Meaning
0b0	Secure Group 1 interrupts are disabled.
0b1	Secure Group 1 interrupts are enabled.

The Secure ICC_IGRPEN1.Enable bit is a read/write alias of the ICC_MGRPEN1.EnableGrp1S bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

EnableGrp1NS, bit [0]

Enables Group 1 interrupts for the Non-secure state.

EnableGrp1NS	Meaning
0b0	Non-secure Group 1 interrupts are disabled.
0b1	Non-secure Group 1 interrupts are enabled.

The Non-secure ICC_IGRPEN1.Enable bit is a read/write alias of the ICC_MGRPEN1.EnableGrp1NS bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICC_MGRPEN1

If an interrupt is pending within the CPU interface when an Enable bit becomes 0, the interrupt must be released to allow the Distributor to forward the interrupt to a different PE.

This register is only accessible when executing in Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b110	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_AA32EL3) && IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICC_MGRPEN1;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b110	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_AA32EL3) && IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_MGRPEN1 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_MSRE, Interrupt Controller Monitor System Register Enable register

The ICC_MSRE characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL3.

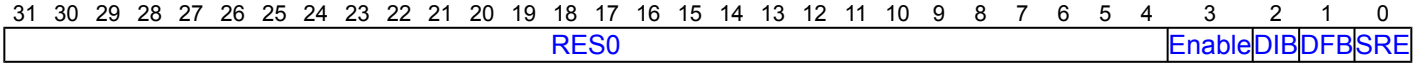
Configuration

This register is present only when FEAT_AA32EL3 is implemented, GICv3 is implemented, and EL3 is implemented. Otherwise, direct accesses to ICC_MSRE are UNDEFINED.

Attributes

ICC_MSRE is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

Enable, bit [3]

Enable. Enables lower Exception level access to ICC_SRE and ICC_HSRE.

Enable	Meaning
0b0	Secure EL1 accesses to Secure ICC_SRE trap to EL3. EL2 accesses to Non-secure ICC_SRE and ICC_HSRE trap to EL3. Non-secure EL1 accesses to ICC_SRE trap to EL3, unless these accesses are trapped to EL2 as a result of ICC_HSRE.Enable == 0.
0b1	Secure EL1 accesses to Secure ICC_SRE do not trap to EL3. EL2 accesses to Non-secure ICC_SRE and ICC_HSRE do not trap to EL3. Non-secure EL1 accesses to ICC_SRE do not trap to EL3.

If ICC_MSRE.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_MSRE.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIB, bit [2]

Disable IRQ bypass.

DIB	Meaning
0b0	IRQ bypass enabled.
0b1	IRQ bypass disabled.

In systems that do not support IRQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

DFB, bit [1]

Disable FIQ bypass.

DFB	Meaning
0b0	FIQ bypass enabled.
0b1	FIQ bypass disabled.

In systems that do not support FIQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

SRE, bit [0]

System Register Enable.

SRE	Meaning
0b0	The memory-mapped interface must be used. Accesses at EL3 or below to any ICH_* System register, or any EL1, EL2, or EL3 ICC_* register other than ICC_SRE, ICC_HSRE, or ICC_MSRE, are UNDEFINED.
0b1	The System register interface to the ICH_* registers and the EL1, EL2, and EL3 ICC_* registers is enabled for EL3.

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICC_MSRE

This register is always System register accessible.

The GIC architecture permits, but does not require, that registers can be shared between memory-mapped registers and the equivalent System registers. This means that if the memory-mapped registers have been accessed while ICC_MSRE.SRE==0, then the System registers might be modified. Therefore, software must only rely on the reset values of the System registers if there has been no use of the GIC functionality while the memory-mapped registers are in use. Otherwise, the System register values must be treated as UNKNOWN.

This register is only accessible when executing in Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b110	0b1100	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_AA32EL3) && IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    R[t] = ICC_MSRE;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b110	0b1100	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_AA32EL3) && IsFeatureImplemented(FEAT_GICv3) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        ICC_MSRE = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_PMR, Interrupt Controller Interrupt Priority Mask Register

The ICC_PMR characteristics are:

Purpose

Provides an interrupt priority filter. Only interrupts with a higher priority than the value in this register are signaled to the PE.

Configuration

AArch32 System register ICC_PMR bits [31:0] are architecturally mapped to AArch64 System register ICC_PMR_EL1[31:0].

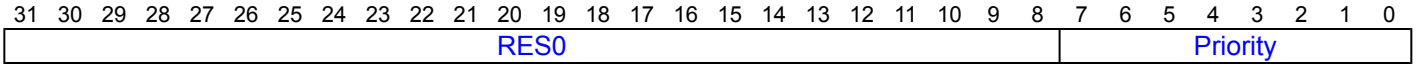
This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_PMR are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronizing. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_PMR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the CPU interface. If the priority of an interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

The possible priority field values are as follows:

Implemented priority bits	Possible priority field values	Number of priority levels
[7:0]	`0x00-0xFF` (0-255), all values	256
[7:1]	`0x00-0xFE` (0-254), even values only	128
[7:2]	`0x00-0xFC` (0-252), in steps of 4	64
[7:3]	`0x00-0xF8` (0-248), in steps of 8	32
[7:4]	`0x00-0xF0` (0-240), in steps of 16	16

Unimplemented priority bits are RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'00000000'`.

Accessing ICC_PMR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0100	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_PMR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_PMR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_PMR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_PMR;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_PMR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_PMR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICC_PMR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0100	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_PMR = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_PMR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_PMR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_PMR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_RPR, Interrupt Controller Running Priority Register

The ICC_RPR characteristics are:

Purpose

Indicates the Running priority of the CPU interface.

Configuration

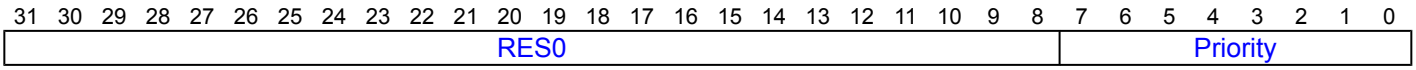
AArch32 System register ICC_RPR bits [31:0] performs the same function as AArch64 System register ICC_RPR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_RPR are UNDEFINED.

Attributes

ICC_RPR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the CPU interface. This is the group priority of the current active interrupt.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

Note

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

Accessing ICC_RPR

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_RPR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_RPR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_RPR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_RPR;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_RPR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_RPR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICC_RPR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SGI0R, Interrupt Controller Software Generated Interrupt Group 0 Register

The ICC_SGI0R characteristics are:

Purpose

Generates Secure Group 0 SGIs.

Configuration

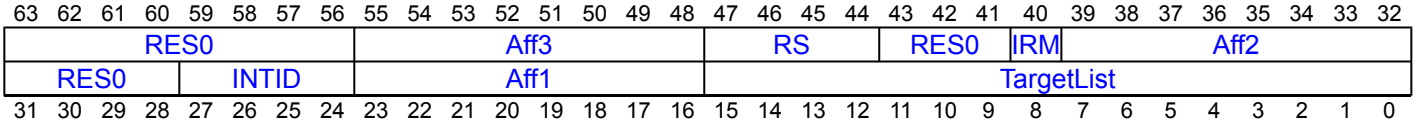
AArch32 System register ICC_SGI0R performs the same function as AArch64 System register ICC_SGI0R_EL1.

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_SGI0R are UNDEFINED.

Attributes

ICC_SGI0R is a 64-bit register.

Field descriptions



Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value ((RS * 16) + n).

When ICC_CTLR_EL1.RSS==0, RS is RES0.

When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a CONSTRAINED UNPREDICTABLE choice of:

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

IRM	Meaning
0b0	Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>.
0b1	Interrupts routed to all PEs in the system, excluding "self".

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

Note

If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Accessing ICC_SGI0R

This register allows software executing in a Secure state to generate Group 0 SGIs. It will also allow software executing in a Non-secure state to generate Group 0 SGIs, if permitted by the settings of GICR_NSACR in the Redistributor corresponding to the target PE.

When GICD_CTLR.DS==0, Non-secure writes do not generate an interrupt for a target PE if not permitted by the GICR_NSACR register associated with the target PE. For more information, see 'Use of control registers for SGI forwarding' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Accesses from Secure Monitor mode are treated as Secure regardless of the value of SCR.NS.

Accesses to this register use the following encodings in the System register encoding space:

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1100	0b0010

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_SGI0R = R[t, t2];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_SGI0R = R[t, t2];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_SGI0R = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SGI1R, Interrupt Controller Software Generated Interrupt Group 1 Register

The ICC_SGI1R characteristics are:

Purpose

Generates Group 1 SGIs for the current Security state.

Configuration

AArch32 System register ICC_SGI1R performs the same function as AArch64 System register ICC_SGI1R_EL1.

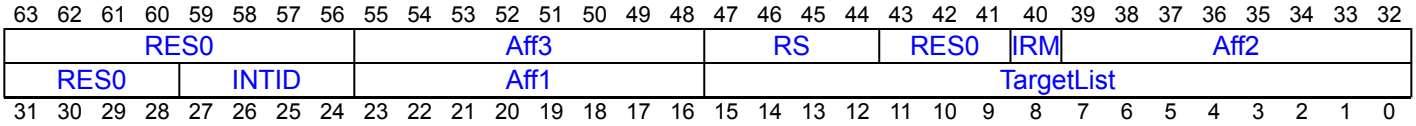
This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_SGI1R are UNDEFINED.

Under certain conditions a write to ICC_SGI1R can generate Group 0 interrupts, see 'Forwarding an SGI to a target PE' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_SGI1R is a 64-bit register.

Field descriptions



Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value ((RS * 16) + n).

When ICC_CTLR_EL1.RSS==0, RS is RES0.

When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a CONSTRAINED UNPREDICTABLE choice of:

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

IRM	Meaning
0b0	Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>.
0b1	Interrupts routed to all PEs in the system, excluding "self".

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

Note

If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Accessing ICC_SGI1R

Note

Accesses from Secure Monitor mode are treated as Secure regardless of the value of SCR.NS.

Accesses to this register use the following encodings in the System register encoding space:

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1100	0b0000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_SGI1R = R[t, t2];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_SGI1R = R[t, t2];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_SGI1R = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SRE, Interrupt Controller System Register Enable register

The ICC_SRE characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL0 and EL1.

Configuration

This register is banked between ICC_SRE and ICC_SRE_S and ICC_SRE_NS.

AArch32 System register ICC_SRE bits [31:0] (ICC_SRE_S) are architecturally mapped to AArch64 System register ICC_SRE_EL1[31:0] (ICC_SRE_EL1_S).

AArch32 System register ICC_SRE bits [31:0] (ICC_SRE_NS) are architecturally mapped to AArch64 System register ICC_SRE_EL1[31:0] (ICC_SRE_EL1_NS).

AArch32 System register ICC_SRE bits [31:0] are architecturally mapped to AArch64 System register ICC_SRE_EL1.

This register is present only when FEAT_AA32EL1 is implemented and GICv3 is implemented. Otherwise, direct accesses to ICC_SRE are UNDEFINED.

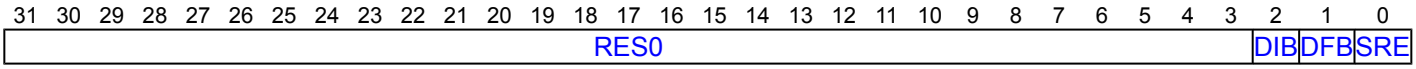
Attributes

ICC_SRE is a 32-bit register.

This register has the following instances:

- ICC_SRE, when EL3 is not implemented or FEAT_AA64 is implemented.
- ICC_SRE_S, when FEAT_AA32EL3 is implemented.
- ICC_SRE_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



Bits [31:3]

Reserved, RES0.

DIB, bit [2]

Disable IRQ bypass.

DIB	Meaning
0b0	IRQ bypass enabled.
0b1	IRQ bypass disabled.

If EL3 is implemented and GICD_CTLR.DS == 0, this field is a read-only alias of ICC_MSRE.DIB.

If EL3 is implemented and GICD_CTLR.DS == 1, and EL2 is not implemented, this field is a read/write alias of ICC_MSRE.DIB.

If EL3 is not implemented and EL2 is implemented, this field is a read-only alias of ICC_HSRE.DIB.

If GICD_CTLR.DS == 1 and EL2 is implemented, this field is a read-only alias of ICC_HSRE.DIB.

In systems that do not support IRQ bypass, this field is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

DFB, bit [1]

Disable FIQ bypass.

DFB	Meaning
0b0	FIQ bypass enabled.
0b1	FIQ bypass disabled.

If EL3 is implemented and GICD_CTLR.DS == 0, this field is a read-only alias of ICC_MSRE.DFB.

If EL3 is implemented and GICD_CTLR.DS == 1, and EL2 is not implemented, this field is a read/write alias of ICC_MSRE.DFB.

If EL3 is not implemented and EL2 is implemented, this field is a read-only alias of ICC_HSRE.DFB.

If GICD_CTLR.DS == 1 and EL2 is implemented, this field is a read-only alias of ICC_HSRE.DFB.

In systems that do not support FIQ bypass, this field is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

SRE, bit [0]

System Register Enable.

SRE	Meaning
0b0	The memory-mapped interface must be used. Accesses at EL1 to any ICC_* System register other than ICC_SRE are UNDEFINED.
0b1	The System register interface for the current Security state is enabled.

If software changes this bit from 1 to 0 in the Secure instance of this register, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and using AArch64:

- When ICC_SRE_EL3.SRE==0 the Secure copy of this bit is RAZ/WI.
- When ICC_SRE_EL3.SRE==0 the Non-secure copy of this bit is RAZ/WI.

If EL3 is implemented and using AArch32:

- When ICC_MSRE.SRE==0 the Secure copy of this bit is RAZ/WI.
- When ICC_MSRE.SRE==0 the Non-secure copy of this bit is RAZ/WI.

If EL2 is implemented and using AArch64:

- When ICC_SRE_EL2.SRE==0 the Non-secure copy of this bit is RAZ/WI.

If EL2 is implemented and using AArch32:

- When ICC_HSRE.SRE==0 the Non-secure copy of this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICC_SRE

The GIC architecture permits, but does not require, that registers can be shared between memory-mapped registers and the equivalent System registers. This means that if the memory-mapped registers have been accessed while ICC_SRE.SRE==0, then the System registers might be modified. Therefore, software must only rely on the reset values of the System registers if there has been no use of the GIC functionality while the memory-mapped registers are in use. Otherwise, the System register values must be treated as UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICC_SRE_EL2.Enable == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICC_HSRE.Enable == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
ICC_MSRE.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            R[t] = ICC_SRE_S;
        else
            R[t] = ICC_SRE_NS;
    else
        R[t] = ICC_SRE;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
ICC_MSRE.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            R[t] = ICC_SRE_S;
        else
            R[t] = ICC_SRE_NS;
    else
        R[t] = ICC_SRE;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        R[t] = ICC_SRE_S;
    else
        R[t] = ICC_SRE_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICC_SRE_EL2.Enable == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICC_HSRE.Enable == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
ICC_MSRE.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_SRE_S = R[t];
        else
            ICC_SRE_NS = R[t];
    else
        ICC_SRE = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
ICC_SRE_EL3.Enable == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
ICC_MSRE.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_SRE_S = R[t];
        else
            ICC_SRE_NS = R[t];
    else
        ICC_SRE = R[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        ICC_SRE_S = R[t];
    else
        ICC_SRE_NS = R[t];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_AP0R<n>, Interrupt Controller Hyp Active Priorities Group 0 Registers, n = 0 - 3

The ICH_AP0R<n> characteristics are:

Purpose

Provides information about Group 0 active priorities for EL2.

Configuration

AArch32 System register ICH_AP0R<n> bits [31:0] are architecturally mapped to AArch64 System register ICH_AP0R<n>_EL2[31:0].

This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_AP0R<n> are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_AP0R<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<x>, bit [x], for x = 31 to 0

Provides the access to the virtual active priorities for Group 0 interrupts. Possible values of each bit are:

P<x>	Meaning
0b0	There is no Group 0 interrupt active at the priority corresponding to that bit.
0b1	There is a Group 0 interrupt active at the priority corresponding to that bit.

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP0R0 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP0R0 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP0R1 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP0R0 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP0R1 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP0R2 in the bits corresponding to 10:Priority[5:1].
- The active state of preemption levels 192 - 254 are held in ICH_AP0R3 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n> and ICH_AP1R<n> might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICH_AP0R<n>

ICH_AP0R1 is implemented only in implementations that support 6 or more bits of preemption. ICH_AP0R2 and ICH_AP0R3 are implemented only in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR.PREbits

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICH_AP0R<n>
- ICH_AP1R<n>

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1000	0b0:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_AP0R[m];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_AP0R[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1000	0b0:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_AP0R[m] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_AP0R[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_AP1R<n>, Interrupt Controller Hyp Active Priorities Group 1 Registers, n = 0 - 3

The ICH_AP1R<n> characteristics are:

Purpose

Provides information about Group 1 active priorities for EL2.

Configuration

AArch32 System register ICH_AP1R<n> bits [31:0] are architecturally mapped to AArch64 System register ICH_AP1R<n>_EL2[31:0].

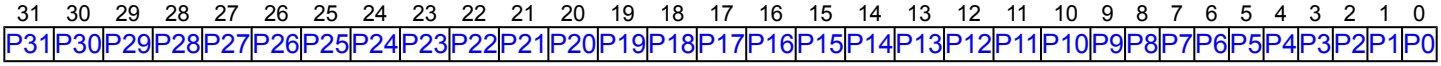
This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_AP1R<n> are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_AP1R<n> is a 32-bit register.

Field descriptions



P<x>, bit [x], for x = 31 to 0

Group 1 interrupt active priorities. Possible values of each bit are:

P<x>	Meaning
0b0	There is no Group 1 interrupt active at the priority corresponding to that bit.
0b1	There is a Group 1 interrupt active at the priority corresponding to that bit.

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP1R0 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP1R0 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP1R1 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP1R0 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP1R1 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP1R2 in the bits corresponding to 10:Priority[5:1].
- The active state of preemption levels 192 - 254 are held in ICH_AP1R3 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n> and ICH_AP1R<n> might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICH_AP1R<n>

ICH_AP1R1 is implemented only in implementations that support 6 or more bits of preemption. ICH_AP1R2 and ICH_AP1R3 are implemented only in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR.PREbits

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICH_AP0R<n>
- ICH_AP1R<n>

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_AP1R[m];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_AP1R[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PREEMPTION_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PREEMPTION_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_AP1R[m] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_AP1R[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_EISR, Interrupt Controller End of Interrupt Status Register

The ICH_EISR characteristics are:

Purpose

Indicates which List registers have outstanding EOI maintenance interrupts.

Configuration

AArch32 System register ICH_EISR bits [31:0] are architecturally mapped to AArch64 System register ICH_EISR_EL2[31:0].

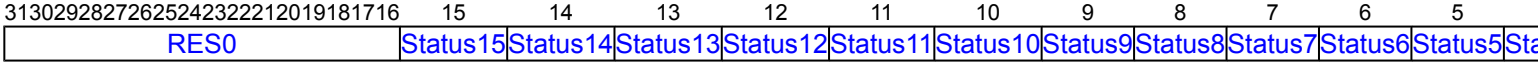
This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_EISR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_EISR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

EOI maintenance interrupt status bit for List register <n>:

Status<n>	Meaning
0b0	List register <n>, ICH_LR<n>, does not have an EOI maintenance interrupt.
0b1	List register <n>, ICH_LR<n>, has an EOI maintenance interrupt that has not been handled.

For any ICH_LR<n>, the corresponding status bit is set to 1 if all of the following are true:

- ICH_LRC<n>.State is 0b00.
- ICH_LRC<n>.HW is 0.
- ICH_LRC<n>.EOI (bit [9]) is 1, indicating that when the interrupt corresponding to that List register is deactivated, a maintenance interrupt is asserted.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICH_EISR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_EISR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_EISR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_ELRSR, Interrupt Controller Empty List Register Status Register

The ICH_ELRSR characteristics are:

Purpose

Indicates which List registers contain valid interrupts.

Configuration

AArch32 System register ICH_ELRSR bits [31:0] are architecturally mapped to AArch64 System register ICH_ELRSR_EL2[31:0].

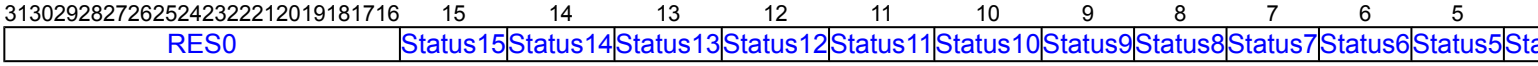
This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_ELRSR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_ELRSR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

Status bit for List register <n>, ICH_LR<n>:

Status<n>	Meaning
0b0	List register ICH_LR<n>, if implemented, contains a valid interrupt. Using this List register can result in overwriting a valid interrupt.
0b1	List register ICH_LR<n> does not contain a valid interrupt. The List register is empty and can be used without overwriting a valid interrupt or losing an EOI maintenance interrupt.

For any List register <n>, the corresponding status bit is set to 1 if ICH_LRC<n>.State is 0b00 and either ICH_LRC<n>.HW is 1 or ICH_LRC<n>.EOI (bit [9]) is 0.

Accessing ICH_ELRSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1011	0b101

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_ELRSR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_ELRSR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_HCR, Interrupt Controller Hyp Control Register

The ICH_HCR characteristics are:

Purpose

Controls the environment for VMs.

Configuration

AArch32 System register ICH_HCR bits [31:0] are architecturally mapped to AArch64 System register ICH_HCR_EL2[31:0].

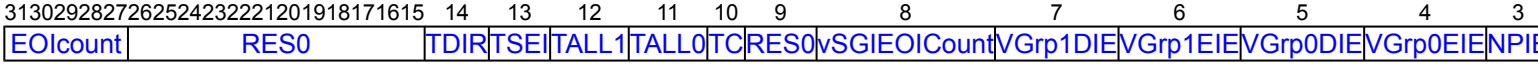
This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_HCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_HCR is a 32-bit register.

Field descriptions



EOIcount, bits [31:27]

This field is incremented whenever a successful write to a virtual EOIR or DIR register would have resulted in a virtual interrupt deactivation. That is either:

- A virtual write to EOIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is zero and no List Register was found.
- A virtual write to DIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is one and no List Register was found.

This allows software to manage more active interrupts than there are implemented List Registers.

It is CONSTRAINED UNPREDICTABLE whether a virtual write to EOIR that does not clear a bit in the Active Priorities registers (ICH_AP0R<n>/ICH_AP1R<n>) increments EOIcount. Permitted behaviors are:

- Increment EOIcount.
- Leave EOIcount unchanged.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'00000'`.

Bits [26:15]

Reserved, RES0.

TDIR, bit [14]

When FEAT_GICv3_TDIR is implemented:

Trap Non-secure EL1 writes to ICC_DIR and ICV_DIR.

TDIR	Meaning
0b0	Non-secure EL1 writes of ICC_DIR and ICV_DIR are not trapped to EL2, unless trapped by other mechanisms.
0b1	Non-secure EL1 writes of ICV_DIR are trapped to EL2. It is IMPLEMENTATION DEFINED whether Non-secure writes of ICC_DIR are trapped. Not trapping ICC_DIR writes is DEPRECATED.

Arm deprecates not including this trap bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

TSEI, bit [13]

Trap all locally generated SEIs. This bit allows the hypervisor to intercept locally generated SEIs that would otherwise be taken at Non-secure EL1.

TSEI	Meaning
0b0	Locally generated SEIs do not cause a trap to EL2.
0b1	Locally generated SEIs trap to EL2.

If ICH_VTR.SEIS is 0, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TALL1, bit [12]

Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 1 interrupts to EL2.

TALL1	Meaning
0b0	Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts proceed as normal.
0b1	Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts trap to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TALL0, bit [11]

Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 0 interrupts to EL2.

TALL0	Meaning
0b0	Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts proceed as normal.
0b1	Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts trap to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TC, bit [10]

Trap all Non-secure EL1 accesses to System registers that are common to Group 0 and Group 1 to EL2.

TC	Meaning
0b0	Non-secure EL1 accesses to common registers proceed as normal.
0b1	Non-secure EL1 accesses to common registers trap to EL2.

This affects accesses to ICC_SGI0R, ICC_SGI1R, ICC_ASGI1R, ICC_CTLR, ICC_DIR, ICC_PMR, ICC_RPR, ICV_CTLR, ICV_DIR, ICV_PMR, and ICV_RPR.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [9]

Reserved, RES0.

vSGIEOICount, bit [8]

When GICv4.1 is implemented:

Controls whether deactivation of virtual SGIs can increment ICH_HCR_EL2.EOIcount

vSGIEOICount	Meaning
0b0	Deactivation of virtual SGIs can increment ICH_HCR.EOIcount.
0b1	Deactivation of virtual SGIs does not increment ICH_HCR.EOIcount.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

VGrp1DIE, bit [7]

VM Group 1 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is disabled:

VGrp1DIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when ICH_VMCR.VENG1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp1EIE, bit [6]

VM Group 1 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is enabled:

VGrp1EIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when ICH_VMCR.VENG1 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp0DIE, bit [5]

VM Group 0 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is disabled:

VGrp0DIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when ICH_VMCR.VENG0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp0EIE, bit [4]

VM Group 0 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is enabled:

VGrp0EIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when ICH_VMCR.VENG0 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

NPIE, bit [3]

No Pending Interrupt Enable. Enables the signaling of a maintenance interrupt when there are no List registers with the State field set to 0b01 (pending):

NPIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled while the List registers contain no interrupts in the pending state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

LRENPIE, bit [2]

List Register Entry Not Present Interrupt Enable. Enables the signaling of a maintenance interrupt while the virtual CPU interface does not have a corresponding valid List register entry for an EOI request:

LRENPIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt is asserted while the EOIcount field is not 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

UIE, bit [1]

Underflow Interrupt Enable. Enables the signaling of a maintenance interrupt when the List registers are empty, or hold only one valid entry:

UIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt is asserted if none, or only one, of the List register entries is marked as a valid interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

En, bit [0]

Enable. Global enable bit for the virtual CPU interface:

En	Meaning
0b0	Virtual CPU interface operation disabled.
0b1	Virtual CPU interface operation enabled.

When this field is set to 0:

- The virtual CPU interface does not signal any maintenance interrupts.
- The virtual CPU interface does not signal any virtual interrupts.
- A read of ICV_IAR0, ICV_IAR1, GICV_IAR or GICV_AIAR returns a spurious interrupt ID.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICH_HCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1011	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_HCR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_HCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1011	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_HCR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_HCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_LRC<n>, Interrupt Controller List Registers, n = 0 - 15

The ICH_LRC<n> characteristics are:

Purpose

Provides interrupt context information for the virtual CPU interface.

Configuration

AArch32 System register ICH_LRC<n> bits [31:0] are architecturally mapped to AArch64 System register ICH_LR<n>_EL2[63:32].

This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_LRC<n> are UNDEFINED.

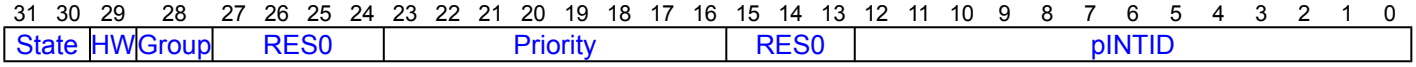
If EL2 is not implemented, this register is RES0 from EL3.

If list register n is not implemented, then accesses to this register are UNDEFINED.

Attributes

ICH_LRC<n> is a 32-bit register.

Field descriptions



State, bits [31:30]

The state of the interrupt:

State	Meaning
0b00	Invalid (Inactive).
0b01	Pending.
0b10	Active.
0b11	Pending and active.

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. Entries in the invalid state are ignored, except for the purpose of generating virtual maintenance interrupts.

For hardware interrupts, the pending and active state is held in the physical Distributor rather than the virtual CPU interface. A hypervisor must only use the pending and active state for software originated interrupts, which are typically associated with virtual devices, or SGIs.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

HW, bit [29]

Indicates whether this virtual interrupt maps directly to a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt with the INTID that the pINTID field indicates.

HW	Meaning
0b0	The interrupt is triggered entirely by software. No notification is sent to the Distributor when the virtual interrupt is deactivated.
0b1	The interrupt maps directly to a hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using the pINTID field from this register to indicate the physical INTID. If ICH_VMCR.VEOIM is 0, this request corresponds to a write to ICC_EOIR0 or ICC_EOIR1. Otherwise, it corresponds to a write to ICC_DIR.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Group, bit [28]

Indicates the group for this virtual interrupt.

Group	Meaning
0b0	This is a Group 0 virtual interrupt. ICH_VMCR.VFIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and ICH_VMCR.VENG0 enables signaling of this interrupt to the virtual machine.
0b1	This is a Group 1 virtual interrupt, signaled as a virtual IRQ. ICH_VMCR.VENG1 enables the signaling of this interrupt to the virtual machine. If ICH_VMCR.VCBPR is 0, then ICC_BPR1 determines if a pending Group 1 interrupt has sufficient priority to preempt current execution. Otherwise, ICH_LR<n> determines preemption.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [27:24]

Reserved, RES0.

Priority, bits [23:16]

The priority of this interrupt.

It is IMPLEMENTATION DEFINED how many bits of priority are implemented, though at least five bits must be implemented. Unimplemented bits are RES0 and start from bit[16] up to bit[18]. The number of implemented bits can be discovered from ICH_VTR.PRIbits.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00000000'.

Bits [15:13]

Reserved, RES0.

pINTID, bits [12:0]

Physical INTID, for hardware interrupts.

When ICH_LRC<n>.HW is 0 (there is no corresponding physical interrupt), this field has the following meaning:

- Bits[12:10]: RES0.
- Bit[9]: EOI. If this bit is 1, then when the interrupt identified by vINTID is deactivated, an EOI maintenance interrupt is asserted.
- Bits[8:0]: Reserved, RES0.

When ICH_LRC<n>.HW is 1 (there is a corresponding physical interrupt):

- This field indicates the physical INTID. This field is only required to implement enough bits to hold a valid value for the implemented INTID size. Any unused higher order bits are RES0.

- When ICC_CTLR_EL1.ExtRange is 0, then bits[44:42] of this field are RES0.
- If the value of pINTID is not a valid INTID, behavior is UNPREDICTABLE. If the value of pINTID indicates a PPI, this field applies to the PPI associated with this same physical PE ID as the virtual CPU interface requesting the deactivation.

A hardware physical identifier is only required in List Registers for interrupts that require deactivation. This means only 13 bits of Physical INTID are required, regardless of the number specified by ICC_CTLR.IDbits.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0000000000000'.

Accessing ICH_LRC<n>

ICH_LR<n> and ICH_LRC<n> can be updated independently.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b111:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif m >= NUM_GIC_LIST_REGS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_LRC[m];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_LRC[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b111:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif m >= NUM_GIC_LIST_REGS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LRC[m] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LRC[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_LR<n>, Interrupt Controller List Registers, n = 0 - 15

The ICH_LR<n> characteristics are:

Purpose

Provides interrupt context information for the virtual CPU interface.

Configuration

AArch32 System register ICH_LR<n> bits [31:0] are architecturally mapped to AArch64 System register ICH_LR<n>_EL2[31:0].

This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_LR<n> are UNDEFINED.

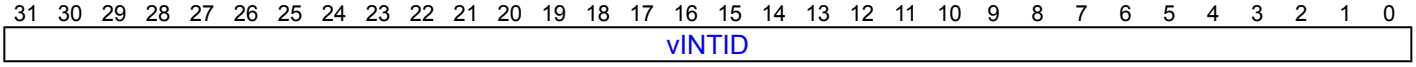
If EL2 is not implemented, this register is RES0 from EL3.

If list register n is not implemented, then accesses to this register are UNDEFINED.

Attributes

ICH_LR<n> is a 32-bit register.

Field descriptions



vINTID, bits [31:0]

Virtual INTID of the interrupt.

If the value of vINTID is 1020-1023 and ICH_LRC<n>.State!=0b00 (Inactive), behavior is UNPREDICTABLE.

Behavior is UNPREDICTABLE if two or more List Registers specify the same vINTID when:

- ICH_LRC<n>.State == 01.
- ICH_LRC<n>.State == 10.
- ICH_LRC<n>.State == 11.

It is IMPLEMENTATION DEFINED how many bits are implemented, though at least 16 bits must be implemented. Unimplemented bits are RES0. The number of implemented bits can be discovered from ICH_VTR.IDbits.

Note

When a VM is using memory-mapped access to the GIC, software must ensure that the correct source PE ID is provided in bits[12:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to '00000000000000000000000000000000'.

Accessing ICH_LR<n>

ICH_LR<n> and ICH_LRC<n> can be updated independently.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b110:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif m >= NUM_GIC_LIST_REGS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_LR[m];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_LR[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-15
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b110:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif m >= NUM_GIC_LIST_REGS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LR[m] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LR[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_MISR, Interrupt Controller Maintenance Interrupt State Register

The ICH_MISR characteristics are:

Purpose

Indicates which maintenance interrupts are asserted.

Configuration

AArch32 System register ICH_MISR bits [31:0] are architecturally mapped to AArch64 System register ICH_MISR_EL2[31:0].

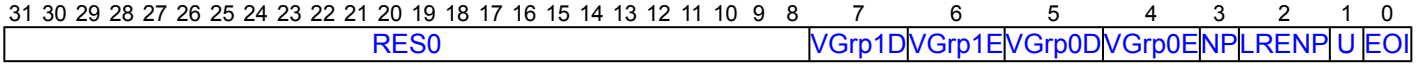
This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_MISR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_MISR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

VGrp1D, bit [7]

vPE Group 1 Disabled.

VGrp1D	Meaning
0b0	vPE Group 1 Disabled maintenance interrupt not asserted.
0b1	vPE Group 1 Disabled maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR.VGrp1DIE is 1 and ICH_VMCR.VENG0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp1E, bit [6]

vPE Group 1 Enabled.

VGrp1E	Meaning
0b0	vPE Group 1 Enabled maintenance interrupt not asserted.
0b1	vPE Group 1 Enabled maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR.VGrp1EIE is 1 and ICH_VMCR.VENG1 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp0D, bit [5]

vPE Group 0 Disabled.

VGrp0D	Meaning
0b0	vPE Group 0 Disabled maintenance interrupt not asserted.
0b1	vPE Group 0 Disabled maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR.VGrp0DIE is 1 and ICH_VMCR.VENG0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp0E, bit [4]

vPE Group 0 Enabled.

VGrp0E	Meaning
0b0	vPE Group 0 Enabled maintenance interrupt not asserted.
0b1	vPE Group 0 Enabled maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR.VGrp0EIE is 1 and ICH_VMCR.VENG0 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

NP, bit [3]

No Pending.

NP	Meaning
0b0	No Pending maintenance interrupt not asserted.
0b1	No Pending maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR.NPIE is 1 and no List register is in pending state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

LRENP, bit [2]

List Register Entry Not Present.

LRENP	Meaning
0b0	List Register Entry Not Present maintenance interrupt not asserted.
0b1	List Register Entry Not Present maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR.LRENPIE is 1 and ICH_HCR.EOIcount is nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

U, bit [1]

Underflow.

U	Meaning
0b0	Underflow maintenance interrupt not asserted.
0b1	Underflow maintenance interrupt asserted.

This maintenance interrupt is asserted when ICH_HCR.UIE is 1 and zero or one of the List register entries are marked as a valid interrupt, that is, if the corresponding ICH_LRC<n>.State bits do not equal 0x0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

EOI, bit [0]

End Of Interrupt.

EOI	Meaning
0b0	End Of Interrupt maintenance interrupt not asserted.
0b1	End Of Interrupt maintenance interrupt asserted.

This maintenance interrupt is asserted when at least one bit in ICH_EISR is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICH_MISR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1011	0b010

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_MISR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_MISR;
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_VMCR, Interrupt Controller Virtual Machine Control Register

The ICH_VMCR characteristics are:

Purpose

Enables the hypervisor to save and restore the virtual machine view of the GIC state.

Configuration

AArch32 System register ICH_VMCR bits [31:0] are architecturally mapped to AArch64 System register ICH_VMCR_EL2[31:0].

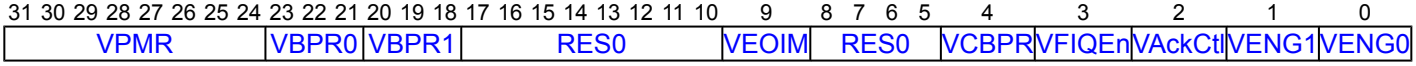
This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_VMCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_VMCR is a 32-bit register.

Field descriptions



VPMR, bits [31:24]

Virtual Priority Mask. The priority mask level for the virtual CPU interface. If the priority of a pending virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

This field is an alias of ICV_PMR.Priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VBPR0, bits [23:21]

Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if ICH_VMCR.VCBPR == 1.

This field is an alias of ICV_BPR0.BinaryPoint.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VBPR1, bits [20:18]

Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption if ICH_VMCR.VCBPR == 0.

This field is an alias of ICV_BPR1.BinaryPoint.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [17:10]

Reserved, RES0.

VEOIM, bit [9]

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

VEOIM	Meaning
0b0	ICV_EOIR0 and ICV_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR are UNPREDICTABLE.
0b1	ICV_EOIR0 and ICV_EOIR1 provide priority drop functionality only. ICV_DIR provides interrupt deactivation functionality.

This bit is an alias of ICV_CTLR.EOImode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:5]

Reserved, RES0.

VCBPR, bit [4]

Virtual Common Binary Point Register. Possible values of this bit are:

VCBPR	Meaning
0b0	ICV_BPR0 determines the preemption group for virtual Group 0 interrupts only. ICV_BPR1 determines the preemption group for virtual Group 1 interrupts.
0b1	ICV_BPR0 determines the preemption group for both virtual Group 0 and virtual Group 1 interrupts. Reads of ICV_BPR1 return ICV_BPR0 plus one, saturated to 0b111. Writes to ICV_BPR1 are ignored.

This field is an alias of ICV_CTLR.CBPR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VFIQEn, bit [3]

Virtual FIQ enable. Possible values of this bit are:

VFIQEn	Meaning
0b0	Group 0 virtual interrupts are presented as virtual IRQs.
0b1	Group 0 virtual interrupts are presented as virtual FIQs.

This bit is an alias of GICV_CTLR.FIQEn.

In implementations where the Non-secure copy of ICC_SRE.SRE is always 1, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VAckCtl, bit [2]

Virtual AckCtl. Possible values of this bit are:

VAckCtl	Meaning
0b0	If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an INTID of 1022.
0b1	If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the INTID of the corresponding interrupt.

This bit is an alias of GICV_CTLR.AckCtl.

This field is supported for backwards compatibility with GICv2. Arm deprecates the use of this field.

In implementations where the Non-secure copy of ICC_SRE.SRE is always 1, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VENG1, bit [1]

Virtual Group 1 interrupt enable. Possible values of this bit are:

VENG1	Meaning
0b0	Virtual Group 1 interrupts are disabled.
0b1	Virtual Group 1 interrupts are enabled.

This bit is an alias of ICV_IGRPEN1.Enable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VENG0, bit [0]

Virtual Group 0 interrupt enable. Possible values of this bit are:

VENG0	Meaning
0b0	Virtual Group 0 interrupts are disabled.
0b1	Virtual Group 0 interrupts are enabled.

This bit is an alias of ICV_IGRPEN0.Enable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICH_VMCR

When EL2 is using System register access, EL1 using either System register or memory-mapped access must be supported.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1011	0b111

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_VMCR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_VMCR;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1011	0b111

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_VMCR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_VMCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_VTR, Interrupt Controller VGIC Type Register

The ICH_VTR characteristics are:

Purpose

Reports supported GIC virtualization features.

Configuration

AArch32 System register ICH_VTR bits [31:0] are architecturally mapped to AArch64 System register ICH_VTR_EL2[31:0].

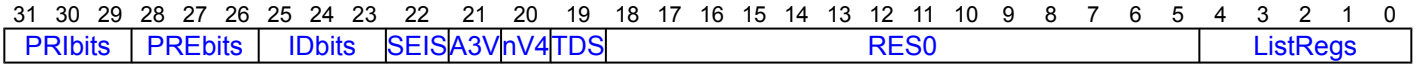
This register is present only when FEAT_AA32EL2 is implemented, GICv3 is implemented, and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_VTR are UNDEFINED.

If EL2 is not implemented, all bits in this register are RES0 from EL3, except for nV4, which is RES1 from EL3.

Attributes

ICH_VTR is a 32-bit register.

Field descriptions



PRIbits, bits [31:29]

Priority bits. The number of virtual priority bits implemented, minus one.

An implementation must implement at least 32 levels of virtual priority (5 priority bits).

This field is an alias of ICV_CTLR.PRIbits.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PREbits, bits [28:26]

The number of virtual preemption bits implemented, minus one.

An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).

The value of this field must be less than or equal to the value of ICH_VTR.PRIbits.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

IDbits, bits [25:23]

The number of virtual interrupt identifier bits supported:

The value of this field is an IMPLEMENTATION DEFINED choice of:

IDbits	Meaning
0b000	16 bits.
0b001	24 bits.

All other values are reserved.

This field is an alias of ICV_CTLR.IDbits.

Access to this field is **RO**.

SEIS, bit [22]

SEI Support. Indicates whether the virtual CPU interface supports generation of SEIs:

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEIS	Meaning
0b0	The virtual CPU interface logic does not support generation of SEIs.
0b1	The virtual CPU interface logic supports generation of SEIs.

This bit is an alias of ICV_CTLR.SEIS.

Access to this field is **RO**.

A3V, bit [21]

Affinity 3 Valid. Possible values are:

The value of this field is an IMPLEMENTATION DEFINED choice of:

A3V	Meaning
0b0	The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.
0b1	The virtual CPU interface logic supports nonzero values of Affinity 3 in SGI generation System registers.

This bit is an alias of ICV_CTLR.A3V.

Access to this field is **RO**.

nV4, bit [20]

Direct injection of virtual interrupts not supported. Possible values are:

The value of this field is an IMPLEMENTATION DEFINED choice of:

nV4	Meaning
0b0	The CPU interface logic supports direct injection of virtual interrupts.
0b1	The CPU interface logic does not support direct injection of virtual interrupts.

In GICv3, the only permitted value is 0b1.

Access to this field is **RO**.

TDS, bit [19]

Separate trapping of Non-secure EL1 writes to ICV_DIR supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TDS	Meaning
0b0	Implementation does not support ICH_HCR.TDIR.
0b1	Implementation supports ICH_HCR.TDIR.

FEAT_GICv3_TDIR implements the functionality added by the value 0b1.

Access to this field is **RO**.

Bits [18:5]

Reserved, RES0.

ListRegs, bits [4:0]

The number of implemented List registers, minus one. For example, a value of `0b01111` indicates that the maximum of 16 List registers are implemented.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ICH_VTR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b1011	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL2) && IsFeatureImplemented(FEAT_GICv3) && (HaveEL(EL2) ||
HaveEL(EL3))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_VTR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICH_VTR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICIALLU, Instruction Cache Invalidate All to PoU

The ICIALLU characteristics are:

Purpose

Invalidate all instruction caches of the PE executing the instruction to the Point of Unification. If branch predictors are architecturally visible, also flush branch predictors.

Configuration

AArch32 System instruction ICIALLU performs the same function as AArch64 System instruction IC IALLU.

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ICIALLU are UNDEFINED.

Attributes

ICIALLU is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing ICIALLU

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

When HCR.FB is 1, at Non-secure EL1 this instruction executes as a ICIALLUIS.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TOCU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TPU ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TOCU
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FB ==
'1' then
        AArch32.IC(CacheOpScope_ALLUIS);
    else
        AArch32.IC(CacheOpScope_ALLU);
elsif PSTATE.EL == EL2 then
    AArch32.IC(CacheOpScope_ALLU);
elsif PSTATE.EL == EL3 then
    AArch32.IC(CacheOpScope_ALLU);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICIALLUIS, Instruction Cache Invalidate All to PoU, Inner Shareable

The ICIALLUIS characteristics are:

Purpose

Invalidate all instruction caches in the Inner Shareable domain of the PE executing the instruction to the Point of Unification. If branch predictors are architecturally visible, also flush branch predictors.

Configuration

AArch32 System instruction ICIALLUIS performs the same function as AArch64 System instruction IC IALLUIS.

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ICIALLUIS are UNDEFINED.

Attributes

ICIALLUIS is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing ICIALLUIS

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TICAB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TPU ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TICAB
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.IC(CacheOpScope_ALLUIS);
elsif PSTATE.EL == EL2 then
    AArch32.IC(CacheOpScope_ALLUIS);
elsif PSTATE.EL == EL3 then
    AArch32.IC(CacheOpScope_ALLUIS);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICIMVAU, Instruction Cache line Invalidate by VA to PoU

The ICIMVAU characteristics are:

Purpose

Invalidate instruction cache line by virtual address to PoU.

Configuration

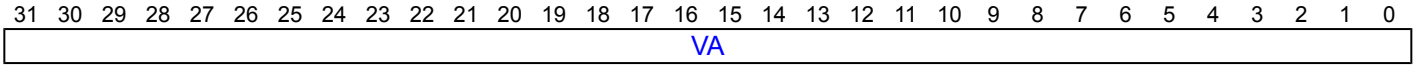
AArch32 System instruction ICIMVAU performs the same function as AArch64 System instruction IC IVAU.

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ICIMVAU are UNDEFINED.

Attributes

ICIMVAU is a 32-bit System instruction.

Field descriptions



VA, bits [31:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing ICIMVAU

Execution of this instruction might require an address translation from VA to PA, and that translation might fault.

For more information about faults, see 'Permission fault'.

For more information about data cache maintenance instructions, see 'AArch32 instruction cache maintenance instructions (IC*)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TOCU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TPU ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TOCU
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.IC(R[t], CacheOpScope_PoU);
elsif PSTATE.EL == EL2 then
    AArch32.IC(R[t], CacheOpScope_PoU);
elsif PSTATE.EL == EL3 then
    AArch32.IC(R[t], CacheOpScope_PoU);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_AP0R<n>, Interrupt Controller Virtual Active Priorities Group 0 Registers, n = 0 - 3

The ICV_AP0R<n> characteristics are:

Purpose

Provides information about virtual Group 0 active priorities.

Configuration

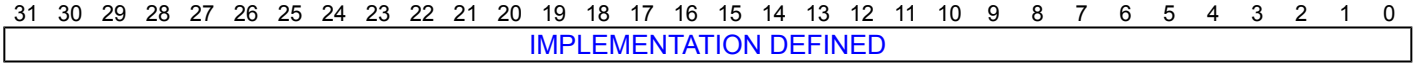
AArch32 System register ICV_AP0R<n> bits [31:0] are architecturally mapped to AArch64 System register ICV_AP0R<n>_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_AP0R<n> are UNDEFINED.

Attributes

ICV_AP0R<n> is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00000000000000000000000000000000'.

Additional information

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICV_AP0R<n>

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 0 active priorities) might result in UNPREDICTABLE behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP0R1 is implemented only in implementations that support 6 or more bits of priority. ICV_AP0R2 and ICV_AP0R3 are implemented only in implementations that support 7 bits of priority. Unimplemented registers are UNDEFINED.

Writing to the active priority registers in any order other than the following order might result in UNPREDICTABLE behavior of the interrupt prioritization system:

- ICV_AP0R<n>.
- ICV_AP1R<n>.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b1:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_AP0R[m];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_AP0R[m];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_AP0R[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_AP0R[m];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICC_AP0R[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b1:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_AP0R[m] = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_AP0R[m] = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_AP0R[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_AP0R[m] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_AP0R[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_AP1R<n>, Interrupt Controller Virtual Active Priorities Group 1 Registers, n = 0 - 3

The ICV_AP1R<n> characteristics are:

Purpose

Provides information about virtual Group 1 active priorities.

Configuration

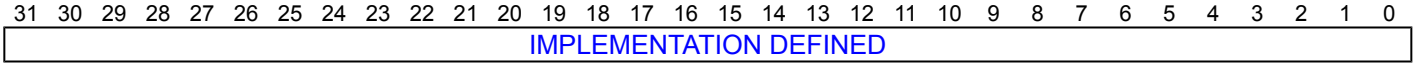
AArch32 System register ICV_AP1R<n> bits [31:0] are architecturally mapped to AArch64 System register ICV_AP1R<n>_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_AP1R<n> are UNDEFINED.

Attributes

ICV_AP1R<n> is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00000000000000000000000000000000'.

Additional information

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICV_AP1R<n>

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP1R1 is implemented only in implementations that support 6 or more bits of priority. ICV_AP1R2 and ICV_AP1R3 are implemented only in implementations that support 7 bits of priority. Unimplemented registers are UNDEFINED.

Writing to the active priority registers in any order other than the following order might result in UNPREDICTABLE behavior of the interrupt prioritization system:

- ICV_AP0R<n>.
- ICV_AP1R<n>.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_AP1R[m];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_AP1R[m];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_AP1R_NS[m];
    else
        R[t] = ICC_AP1R[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_AP1R_NS[m];
    else
        R[t] = ICC_AP1R[m];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            R[t] = ICC_AP1R_S[m];
        else
            R[t] = ICC_AP1R_NS[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-3
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1001	0b0:m[1:0]

```
integer m = UInt(opc2<1:0>);

if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif m == 1 && NUM_GIC_PRIORITY_BITS < 6 then
    UNDEFINED;
elsif (m == 2 || m == 3) && NUM_GIC_PRIORITY_BITS < 7 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_AP1R[m] = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_AP1R[m] = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_AP1R_NS[m] = R[t];
    else
        ICC_AP1R[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_AP1R_NS[m] = R[t];
    else
        ICC_AP1R[m] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            ICC_AP1R_S[m] = R[t];
        else
            ICC_AP1R_NS[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_BPR0, Interrupt Controller Virtual Binary Point Register 0

The ICV_BPR0 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 0 interrupt preemption.

Configuration

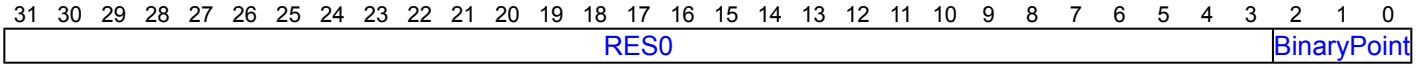
AArch32 System register ICV_BPR0 bits [31:0] are architecturally mapped to AArch64 System register ICV_BPR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_BPR0 are UNDEFINED.

Attributes

ICV_BPR0 is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

Binary point value	Group priority field	Subpriority field	Field with binary point
0	[7:1]	[0]	ggggggg.s
1	[7:2]	[1:0]	gggggg.ss
2	[7:3]	[2:0]	ggggg.sss
3	[7:4]	[3:0]	gggg.ssss
4	[7:5]	[4:0]	ggg.sssss
5	[7:6]	[5:0]	gg.ssssss
6	[7]	[6:0]	g.sssssss
7	No preemption	[7:0]	.sssssss

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_BPR0

The minimum binary point value is derived from the number of implemented priority bits. The number of priority bits is IMPLEMENTATION DEFINED, and reported by ICV_CTLR.PRIbits.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is set to the minimum supported value.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_BPR0;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_BPR0;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_BPR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_BPR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_BPR0;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_BPR0 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_BPR0 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_BPR0 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_BPR0 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        ICC_BPR0 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_BPR1, Interrupt Controller Virtual Binary Point Register 1

The ICV_BPR1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 1 interrupt preemption.

Configuration

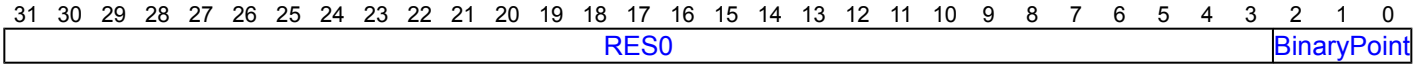
AArch32 System register ICV_BPR1 bits [31:0] are architecturally mapped to AArch64 System register ICV_BPR1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_BPR1 are UNDEFINED.

Attributes

ICV_BPR1 is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

If the GIC is configured to use separate binary point fields for Group 0 and Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

For more information about priorities, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

An attempt to program this field to a value less than the minimum value sets the field to the minimum value.

If ICV_CTLR.CBPR is set to 1, Non-secure EL1 reads return ICV_BPR0 + 1 saturated to `0b111`. Non-secure EL1 writes are ignored.

If ICV_CTLR.CBPR is set to 1, Secure EL1 reads return ICV_BPR0. Secure EL1 writes modify ICV_BPR0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_BPR1

The minimum value of this register is equal to the minimum value of ICV_BPR0 plus one.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_BPR1;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_BPR1;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_BPR1_NS;
    else
        R[t] = ICC_BPR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_BPR1_NS;
```

```
    else
        R[t] = ICC_BPR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            R[t] = ICC_BPR1_S;
        else
            R[t] = ICC_BPR1_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_BPR1 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_BPR1 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_BPR1_NS = R[t];
    else
        ICC_BPR1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_BPR1_NS = R[t];
```

```
    else
        ICC_BPR1 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            ICC_BPR1_S = R[t];
        else
            ICC_BPR1_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_CTLR, Interrupt Controller Virtual Control Register

The ICV_CTLR characteristics are:

Purpose

Controls aspects of the behavior of the GIC virtual CPU interface and provides information about the features implemented.

Configuration

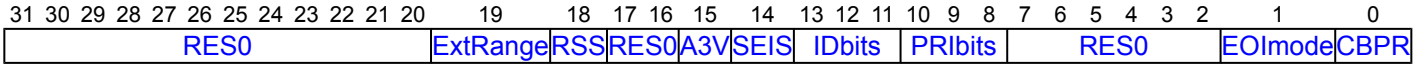
AArch32 System register ICV_CTLR bits [31:0] are architecturally mapped to AArch64 System register ICV_CTLR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_CTLR are UNDEFINED.

Attributes

ICV_CTLR is a 32-bit register.

Field descriptions



Bits [31:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

ExtRange	Meaning
0b0	CPU interface does not support INTIDs in the range 1024..8191. Behavior is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface. **Note** Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.
0b1	CPU interface supports INTIDs in the range 1024..8191. All INTIDs in the range 1024..8191 are treated as requiring deactivation.

ICV_CTLR.ExtRange is an alias of ICC_CTLR.ExtRange.

RSS, bit [18]

Range Selector Support. Possible values are:

RSS	Meaning
0b0	Targeted SGIs with affinity level 0 values of 0 - 15 are supported.
0b1	Targeted SGIs with affinity level 0 values of 0 - 255 are supported.

This bit is read-only.

Bits [17:16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

A3V	Meaning
0b0	The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.
0b1	The virtual CPU interface logic supports nonzero values of Affinity 3 in SGI generation System registers.

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the virtual CPU interface supports local generation of SEIs:

SEIS	Meaning
0b0	The virtual CPU interface logic does not support local generation of SEIs.
0b1	The virtual CPU interface logic supports local generation of SEIs.

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. The number of virtual interrupt identifier bits supported:

IDbits	Meaning
0b000	16 bits.
0b001	24 bits.

All other values are reserved.

PRIbits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of virtual priority bits implemented, minus one.

An implementation must implement at least 32 levels of virtual priority (5 priority bits).

The division between group priority and subpriority is defined in the binary point registers ICV_BPR0 and ICV_BPR1.

Bits [7:2]

Reserved, RES0.

EOImode, bit [1]

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

EOImode	Meaning
0b0	ICV_EOIR0 and ICV_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR are UNPREDICTABLE.
0b1	ICV_EOIR0 and ICV_EOIR1 provide priority drop functionality only. ICV_DIR provides interrupt deactivation functionality.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR, bit [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both virtual Group 0 and virtual Group 1 interrupts:

CBPR	Meaning
0b0	ICV_BPR0 determines the preemption group for virtual Group 0 interrupts only. ICV_BPR1 determines the preemption group for virtual Group 1 interrupts.
0b1	Non-secure reads of ICV_BPR1 return ICV_BPR0 plus one, saturated to 0b111. Non-secure writes to ICV_BPR1 are ignored. Secure reads of ICV_BPR1 return ICV_BPR0. Secure writes of ICV_BPR1 modify ICV_BPR0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_CTLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_CTLR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_CTLR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_CTLR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_CTLR;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_CTLR_NS;
    else
        R[t] = ICC_CTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_CTLR_NS;
    else
        R[t] = ICC_CTLR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            R[t] = ICC_CTLR_S;
        else
            R[t] = ICC_CTLR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_CTLR = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_CTLR_NS = R[t];
    else
        ICC_CTLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_CTLR_NS = R[t];
    else
        ICC_CTLR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            ICC_CTLR_S = R[t];
        else
            ICC_CTLR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_DIR, Interrupt Controller Deactivate Virtual Interrupt Register

The ICV_DIR characteristics are:

Purpose

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified virtual interrupt.

Configuration

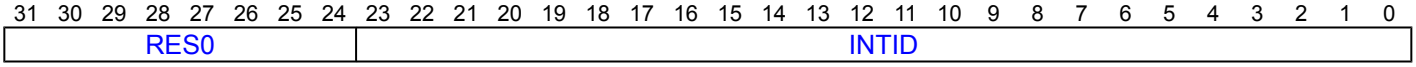
AArch32 System register ICV_DIR bits [31:0] performs the same function as AArch64 System register ICV_DIR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_DIR are UNDEFINED.

Attributes

ICV_DIR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the virtual interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_DIR

When EOImode == 0, writes are ignored. In systems supporting system error generation, an implementation might generate an SEI.

Accesses to this register use the following encodings in the System register encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1011	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TDIR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TDIR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_DIR = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_DIR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_DIR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_DIR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_EOIR0, Interrupt Controller Virtual End Of Interrupt Register 0

The ICV_EOIR0 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 0 interrupt.

Configuration

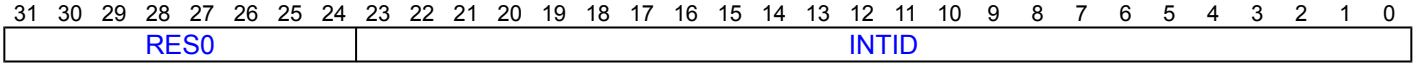
AArch32 System register ICV_EOIR0 bits [31:0] performs the same function as AArch64 System register ICV_EOIR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_EOIR0 are UNDEFINED.

Attributes

ICV_EOIR0 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding ICV_IAR0 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the ICV_CTLR.EOImode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the ICV_CTLR.EOImode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to ICV_DIR to deactivate the virtual interrupt.

Accessing ICV_EOIR0

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICV_IAR0, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings in the System register encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_EOIR0 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_EOIR0 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR0 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR0 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        ICC_EOIR0 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_EOIR1, Interrupt Controller Virtual End Of Interrupt Register 1

The ICV_EOIR1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 1 interrupt.

Configuration

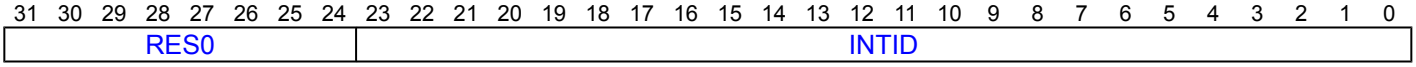
AArch32 System register ICV_EOIR1 bits [31:0] performs the same function as AArch64 System register ICV_EOIR1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_EOIR1 are UNDEFINED.

Attributes

ICV_EOIR1 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding ICV_IAR1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the ICV_CTLR.EOImode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the ICV_CTLR.EOImode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to ICV_DIR to deactivate the virtual interrupt.

Accessing ICV_EOIR1

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICV_IAR1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings in the System register encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_EOIR1 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_EOIR1 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        ICC_EOIR1 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_HPPIR0, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

The ICV_HPPIR0 characteristics are:

Purpose

Indicates the highest priority pending virtual Group 0 interrupt on the virtual CPU interface.

Configuration

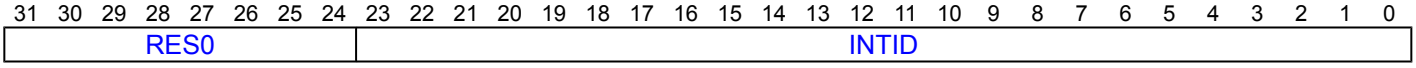
AArch32 System register ICV_HPPIR0 bits [31:0] performs the same function as AArch64 System register ICV_HPPIR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_HPPIR0 are UNDEFINED.

Attributes

ICV_HPPIR0 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_HPPIR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b010

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_HPPIR0;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_HPPIR0;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_HPPIR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_HPPIR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_HPPIR0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_HPPIR1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

The ICV_HPPIR1 characteristics are:

Purpose

Indicates the highest priority pending virtual Group 1 interrupt on the virtual CPU interface.

Configuration

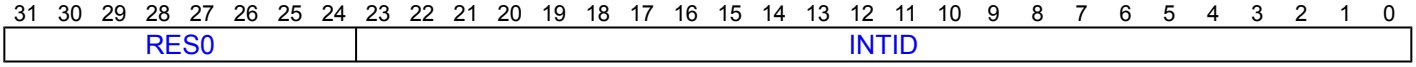
AArch32 System register ICV_HPPIR1 bits [31:0] performs the same function as AArch64 System register ICV_HPPIR1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_HPPIR1 are UNDEFINED.

Attributes

ICV_HPPIR1 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_HPPIR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_HPPIR1;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_HPPIR1;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_HPPIR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_HPPIR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_HPPIR1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IAR0, Interrupt Controller Virtual Interrupt Acknowledge Register 0

The ICV_IAR0 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 0 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch32 System register ICV_IAR0 bits [31:0] performs the same function as AArch64 System register ICV_IAR0_EL1[31:0].

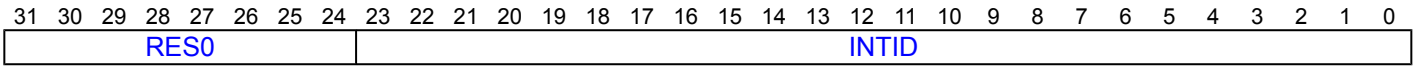
This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_IAR0 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronizing when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_IAR0 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_IAR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_IAR0;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_IAR0;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IAR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IAR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_IAR0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IAR1, Interrupt Controller Virtual Interrupt Acknowledge Register 1

The ICV_IAR1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch32 System register ICV_IAR1 bits [31:0] performs the same function as AArch64 System register ICV_IAR1_EL1[31:0].

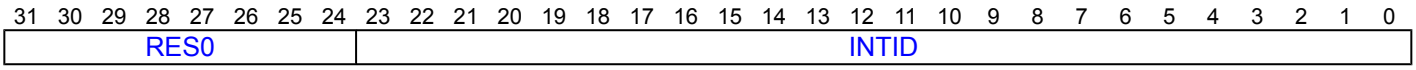
This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_IAR1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronizing when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_IAR1 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_IAR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_IAR1;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_IAR1;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IAR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IAR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_IAR1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IGRPEN0, Interrupt Controller Virtual Interrupt Group 0 Enable register

The ICV_IGRPEN0 characteristics are:

Purpose

Controls whether virtual Group 0 interrupts are enabled or not.

Configuration

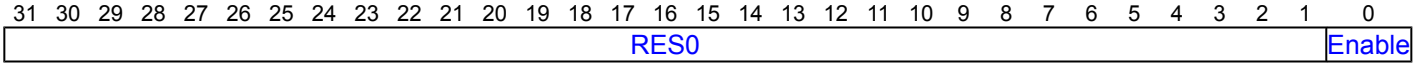
AArch32 System register ICV_IGRPEN0 bits [31:0] are architecturally mapped to AArch64 System register ICV_IGRPEN0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_IGRPEN0 are UNDEFINED.

Attributes

ICV_IGRPEN0 is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

Enable, bit [0]

Enables virtual Group 0 interrupts.

Enable	Meaning
0b0	Virtual Group 0 interrupts are disabled.
0b1	Virtual Group 0 interrupts are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICV_IGRPEN0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_IGRPEN0;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_IGRPEN0;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IGRPEN0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_IGRPEN0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        R[t] = ICC_IGRPEN0;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_IGRPEN0 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_IGRPEN0 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.FIQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_IGRPEN0 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.FIQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_IGRPEN0 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
```

```
        UNDEFINED;
    else
        ICC_IGRPEN0 = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IGRPEN1, Interrupt Controller Virtual Interrupt Group 1 Enable register

The ICV_IGRPEN1 characteristics are:

Purpose

Controls whether virtual Group 1 interrupts are enabled for the current Security state.

Configuration

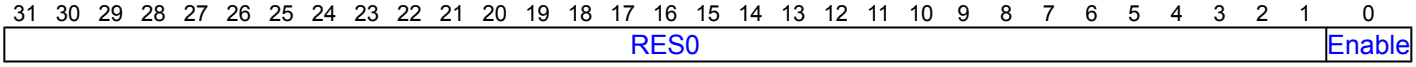
AArch32 System register ICV_IGRPEN1 bits [31:0] are architecturally mapped to AArch64 System register ICV_IGRPEN1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_IGRPEN1 are UNDEFINED.

Attributes

ICV_IGRPEN1 is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

Enable, bit [0]

Enables virtual Group 1 interrupts.

Enable	Meaning
0b0	Virtual Group 1 interrupts are disabled.
0b1	Virtual Group 1 interrupts are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing ICV_IGRPEN1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_IGRPEN1;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_IGRPEN1;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_IGRPEN1_NS;
    else
        R[t] = ICC_IGRPEN1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        R[t] = ICC_IGRPEN1_NS;
```

```
    else
        R[t] = ICC_IGRPEN1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            R[t] = ICC_IGRPEN1_S;
        else
            R[t] = ICC_IGRPEN1_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_IGRPEN1 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_IGRPEN1 = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.IRQ == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_IGRPEN1_NS = R[t];
    else
        ICC_IGRPEN1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ
== '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.IRQ ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_IGRPEN1_NS = R[t];
```

```
    else
        ICC_IGRPEN1 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            ICC_IGRPEN1_S = R[t];
        else
            ICC_IGRPEN1_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_PMR, Interrupt Controller Virtual Interrupt Priority Mask Register

The ICV_PMR characteristics are:

Purpose

Provides a virtual interrupt priority filter. Only virtual interrupts with a higher priority than the value in this register are signaled to the PE.

Configuration

AArch32 System register ICV_PMR bits [31:0] are architecturally mapped to AArch64 System register ICV_PMR_EL1[31:0].

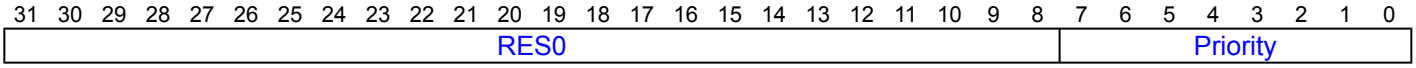
This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_PMR are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronizing. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_PMR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the virtual CPU interface. If the priority of a virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

The possible priority field values are as follows:

Implemented priority bits	Possible priority field values	Number of priority levels
[7:0]	0x00-0xFF (0-255), all values	256
[7:1]	0x00-0xFE (0-254), even values only	128
[7:2]	0x00-0xFC (0-252), in steps of 4	64
[7:3]	0x00-0xF8 (0-248), in steps of 8	32
[7:4]	0x00-0xF0 (0-240), in steps of 16	16

Unimplemented priority bits are RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00000000'.

Accessing ICV_PMR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0100	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_PMR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_PMR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_PMR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_PMR;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_PMR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_PMR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICC_PMR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0100	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        ICV_PMR = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_PMR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_PMR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_PMR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_RPR, Interrupt Controller Virtual Running Priority Register

The ICV_RPR characteristics are:

Purpose

Indicates the Running priority of the virtual CPU interface.

Configuration

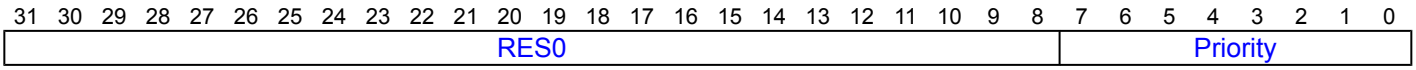
AArch32 System register ICV_RPR bits [31:0] performs the same function as AArch64 System register ICV_RPR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented, GICv3 is implemented, and EL2 is implemented. Otherwise, direct accesses to ICV_RPR are UNDEFINED.

Attributes

ICV_RPR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the virtual CPU interface. This is the group priority of the current active virtual interrupt.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

Note

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

Accessing ICV_RPR

If there are no active interrupts on the virtual CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_GICv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && ICH_HCR.TC
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.FMO == '1' then
        R[t] = ICV_RPR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.IMO == '1' then
        R[t] = ICV_RPR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FMO ==
'1' then
        R[t] = ICV_RPR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.IMO ==
'1' then
        R[t] = ICV_RPR;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_RPR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) &&
SCR.<IRQ,FIQ> == '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = ICC_RPR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        R[t] = ICC_RPR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AFR0, Auxiliary Feature Register 0

The ID_AFR0 characteristics are:

Purpose

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch32 state.

Must be interpreted with the Main ID Register, MIDR.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_AFR0 bits [31:0] are architecturally mapped to AArch64 System register ID_AFR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_AFR0 are UNDEFINED.

Attributes

ID_AFR0 is a 32-bit register.

Field descriptions

Bits [31:16]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [15:12]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [11:8]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [7:4]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [3:0]

IMPLEMENTATION DEFINED.

Accessing ID_AFR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0001	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_AFR0;
elsif PSTATE.EL == EL2 then
    R[t] = ID_AFR0;
elsif PSTATE.EL == EL3 then
    R[t] = ID_AFR0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_DFR0, Debug Feature Register 0

The ID_DFR0 characteristics are:

Purpose

Provides top-level information about the debug system in AArch32 state.

Must be interpreted with the Main ID Register, MIDR.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

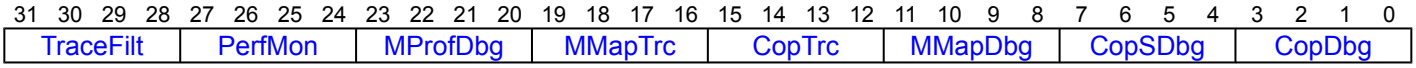
AArch32 System register ID_DFR0 bits [31:0] are architecturally mapped to AArch64 System register ID_DFR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_DFR0 are UNDEFINED.

Attributes

ID_DFR0 is a 32-bit register.

Field descriptions



TraceFilt, bits [31:28]

Armv8.4 Self-hosted Trace Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceFilt	Meaning
0b0000	Armv8.4 Self-hosted Trace Extension not implemented.
0b0001	Armv8.4 Self-hosted Trace Extension implemented.

All other values are reserved.

FEAT_TRF implements the functionality identified by the value 0b0001.

From Armv8.4, if FEAT_ETMv4 is implemented, the value 0b0000 is not permitted.

If FEAT_ETE is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

PerfMon, bits [27:24]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version'.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PerfMon	Meaning
0b0000	Performance Monitors Extension not implemented.
0b0001	Performance Monitors Extension, PMUv1 implemented.
0b0010	Performance Monitors Extension, PMUv2 implemented.
0b0011	Performance Monitors Extension, PMUv3 implemented.
0b0100	PMUv3 for Armv8.1. As 0b0011, and adds support for: • Extended 16-bit PMEVTYPER<n>.evtCount field. • If EL2 is implemented, the HDCR.HPMD control.
0b0101	PMUv3 for Armv8.4. As 0b0100, and adds support for the PMMIR register.
0b0110	PMUv3 for Armv8.5. As 0b0101, and adds support for: • 64-bit event counters. • If EL2 is implemented, the HDCR.HCCD control. • If EL3 is implemented, the SDCR.SCCD control.
0b0111	PMUv3 for Armv8.7. As 0b0110, and adds support for: • The PMCR.FZO and, if EL2 is implemented, HDCR.HPMFZO controls. • If EL3 is implemented and using AArch64, the MDCR_EL3.{MPMX,MCCD} controls.
0b1000	PMUv3 for Armv8.8. As 0b0111, and: • Extends the Common event number space to include 0x0040 to 0x00BF and 0x4040 to 0x40BF. • Removes the CONSTRAINED UNPREDICTABLE behaviors if a reserved or unimplemented PMU event number is selected.
0b1001	PMUv3 for Armv8.9. As 0b1000, and: • Updates the definitions of existing PMU events. • Adds support for the EDECR.PME control.
0b1111	IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value for new implementations.

All other values are reserved.

FEAT_PMUv3 implements the functionality identified by the value 0b0011.

FEAT_PMUv3p1 implements the functionality identified by the value 0b0100.

FEAT_PMUv3p4 implements the functionality identified by the value 0b0101.

FEAT_PMUv3p5 implements the functionality identified by the value 0b0110.

FEAT_PMUv3p7 implements the functionality identified by the value 0b0111.

FEAT_PMUv3p8 implements the functionality identified by the value 0b1000.

FEAT_PMUv3p9 implements the functionality identified by the value 0b1001.

In any Armv8 implementation, the values 0b0001 and 0b0010 are not permitted.

From Armv8.1, if FEAT_PMUv3 is implemented, the value 0b0011 is not permitted.

From Armv8.4, if FEAT_PMUv3 is implemented, the value 0b0100 is not permitted.

From Armv8.5, if FEAT_PMUv3 is implemented, the value 0b0101 is not permitted.

From Armv8.7, if FEAT_PMUv3 is implemented, the value 0b0110 is not permitted.

From Armv8.8, if FEAT_PMUv3 is implemented, the value 0b0111 is not permitted.

From Armv8.9, if FEAT_PMUv3 is implemented, the value 0b1000 is not permitted.

Note

In Armv7, the value 0b0000 can mean that PMUv1 is implemented. PMUv1 and PMUv2 are not permitted in an Armv8 implementation.

Access to this field is **RO**.

MProfDbg, bits [23:20]

M-profile Debug. Support for memory-mapped debug model for M-profile processors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MProfDbg	Meaning
0b0000	Not supported.
0b0001	Support for M-profile Debug architecture, with memory-mapped access.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

MMapTrc, bits [19:16]

Memory-mapped Trace. Support for memory-mapped trace model.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MMapTrc	Meaning
0b0000	Not supported.
0b0001	Support for Arm trace architecture, with memory-mapped access.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

For more information, see the Arm® Embedded Trace Macrocell Architecture Specification, ETMv4 (ARM IHI 0064).

Access to this field is **RO**.

CopTrc, bits [15:12]

Support for System registers-based trace model, using registers in the coproc == 0b1110 encoding space.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CopTrc	Meaning
0b0000	Not supported.
0b0001	Support for Arm trace architecture, with System registers access.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

For more information, see the Arm® Embedded Trace Macrocell Architecture Specification, ETMv4 (ARM IHI 0064).

Access to this field is **RO**.

MMapDbg, bits [11:8]

Memory-mapped Debug. Support for Armv7 memory-mapped debug model for A and R-profile processors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MMapDbg	Meaning
0b0000	Not supported.
0b0100	Support for Armv7, v7 Debug architecture, with memory-mapped access.
0b0101	Support for Armv7, v7.1 Debug architecture, with memory-mapped access.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

The optional memory map defined by Armv8 is not compatible with Armv7.

Access to this field is **RO**.

CopSDbg, bits [7:4]

Support for a System registers-based Secure debug model, using registers in the coproc = 0b1110 encoding space, for an A-profile processor that includes EL3.

If EL3 is not implemented and the implemented Security state is Non-secure state, this field is RES0. Otherwise, this field reads the same as bits [3:0].

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CopDbg, bits [3:0]

Debug architecture version. Indicates presence of Armv8 debug architecture.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CopDbg	Meaning
0b0000	Not supported.
0b0010	Armv6, v6 Debug architecture, with System registers access.
0b0011	Armv6, v6.1 Debug architecture, with System registers access.
0b0100	Armv7, v7 Debug architecture, with System registers access.
0b0101	Armv7, v7.1 Debug architecture, with System registers access.
0b0110	Armv8 debug architecture.
0b0111	Armv8.1 debug architecture, FEAT_Debugv8p1.
0b1000	Armv8.2 debug architecture, FEAT_Debugv8p2.
0b1001	Armv8.4 debug architecture, FEAT_Debugv8p4.
0b1010	Armv8.8 debug architecture, FEAT_Debugv8p8.
0b1011	Armv8.9 debug architecture, FEAT_Debugv8p9.

All other values are reserved.

The values 0b0000, 0b0010, 0b0011, 0b0100, and 0b0101 are not permitted in Armv8.

FEAT_Debugv8p1 implements the functionality identified by the value 0b0111.

FEAT_Debugv8p2 implements the functionality identified by the value 0b1000.

FEAT_Debugv8p4 implements the functionality identified by the value 0b1001.

FEAT_Debugv8p8 implements the functionality identified by the value 0b1010.

FEAT_Debugv8p9 implements the functionality identified by the value 0b1011.

From Armv8.1, when FEAT_Debugv8p1 is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

From Armv8.9, the value 0b1010 is not permitted.

Access to this field is **RO**.

Accessing ID_DFR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_DFR0;
elsif PSTATE.EL == EL2 then
    R[t] = ID_DFR0;
elsif PSTATE.EL == EL3 then
    R[t] = ID_DFR0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_DFR1, Debug Feature Register 1

The ID_DFR1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch32.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_DFR1 bits [31:0] are architecturally mapped to AArch64 System register ID_DFR1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_DFR1 are UNDEFINED.

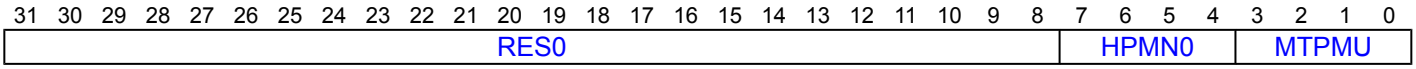
Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_DFR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

HPMN0, bits [7:4]

Zero PMU event counters for a Guest operating system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HPMN0	Meaning
0b0000	Setting HDCR.HPMN to zero has CONSTRAINED UNPREDICTABLE behavior.
0b0001	Setting HDCR.HPMN to zero has defined behavior.

All other values are reserved.

If FEAT_PMUv3 is not implemented, FEAT_FGT is not implemented, or EL2 is not implemented, the only permitted value is 0b0000.

FEAT_HPMN0 implements the functionality identified by the value 0b0001.

From Armv8.8, in an implementation that includes FEAT_PMUv3, FEAT_FGT, and EL2, the value 0b0000 is not permitted.

Access to this field is **RO**.

MTPMU, bits [3:0]

Multi-threaded PMU extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTPMU	Meaning
0b0000	FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, it is IMPLEMENTATION DEFINED whether PMEVTYPER<n>.MT are read/write or RES0.
0b0001	FEAT_MTPMU and FEAT_PMUv3 implemented. PMEVTYPER<n>.MT are read/write. When FEAT_MTPMU is disabled, the Effective values of PMEVTYPER<n>.MT are 0.
0b1111	FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, PMEVTYPER<n>.MT are RES0.

All other values are reserved.

FEAT_MTPMU implements the functionality identified by the value 0b0001.

From Armv8.6, in an implementation that includes FEAT_PMUv3, the value 0b0000 is not permitted.

In an implementation that does not include FEAT_PMUv3, the value 0b0001 is not permitted.

Access to this field is **RO**.

Accessing ID_DFR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0011	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
(IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_DFR1) || boolean IMPLEMENTATION_DEFINED "ID_DFR1
trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
(IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_DFR1) || boolean IMPLEMENTATION_DEFINED "ID_DFR1
trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_DFR1;
elsif PSTATE.EL == EL2 then
    R[t] = ID_DFR1;
elsif PSTATE.EL == EL3 then
    R[t] = ID_DFR1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR0, Instruction Set Attribute Register 0

The ID_ISAR0 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR1, ID_ISAR2, ID_ISAR3, ID_ISAR4, and ID_ISAR5.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

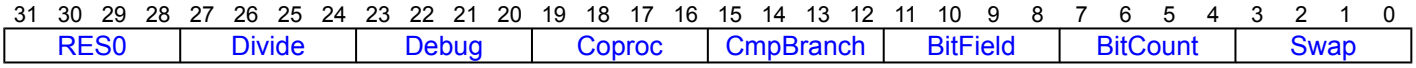
AArch32 System register ID_ISAR0 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_ISAR0 are UNDEFINED.

Attributes

ID_ISAR0 is a 32-bit register.

Field descriptions



Bits [31:28]

Reserved, RES0.

Divide, bits [27:24]

Indicates the implemented Divide instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Divide	Meaning
0b0000	None implemented.
0b0001	Adds SDIV and UDIV in the T32 instruction set.
0b0010	As for 0b0001, and adds SDIV and UDIV in the A32 instruction set.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

Debug, bits [23:20]

Indicates the implemented Debug instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Debug	Meaning
0b0000	None implemented.
0b0001	Adds BKPT.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Coproc, bits [19:16]

Indicates the implemented System register access instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Coproc	Meaning
0b0000	None implemented, except for instructions separately attributed by the architecture to provide access to AArch32 System registers and System instructions.
0b0001	Adds generic CDP, LDC, MCR, MRC, and STC.
0b0010	As for 0b0001, and adds generic CDP2, LDC2, MCR2, MRC2, and STC2.
0b0011	As for 0b0010, and adds generic MCRR and MRRC.
0b0100	As for 0b0011, and adds generic MCRR2 and MRRC2.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

CmpBranch, bits [15:12]

Indicates the implemented combined Compare and Branch instructions in the T32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CmpBranch	Meaning
0b0000	None implemented.
0b0001	Adds CBNZ and CBZ.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

BitField, bits [11:8]

Indicates support for BitField instructions BFC, BFI, SBFX, and UBFX.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BitField	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

BitCount, bits [7:4]

Indicates the implemented Bit Counting instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BitCount	Meaning
0b0000	None implemented.
0b0001	Adds CLZ.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Swap, bits [3:0]

Indicates the implemented Swap instructions in the A32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Swap	Meaning
0b0000	None implemented.
0b0001	Adds SWP and SWPB.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

Accessing ID_ISAR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_ISAR0;
elsif PSTATE.EL == EL2 then
    R[t] = ID_ISAR0;
elsif PSTATE.EL == EL3 then
    R[t] = ID_ISAR0;
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR1, Instruction Set Attribute Register 1

The ID_ISAR1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0, ID_ISAR2, ID_ISAR3, ID_ISAR4, and ID_ISAR5.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

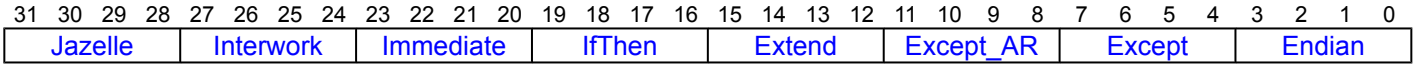
AArch32 System register ID_ISAR1 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_ISAR1 are UNDEFINED.

Attributes

ID_ISAR1 is a 32-bit register.

Field descriptions



Jazelle, bits [31:28]

Indicates the implemented Jazelle extension instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Jazelle	Meaning
0b0000	No support for Jazelle.
0b0001	Adds the BXJ instruction, and the J bit in the PSR. This setting might indicate a trivial implementation of the Jazelle extension.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Interwork, bits [27:24]

Indicates the implemented Interworking instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Interwork	Meaning
0b0000	None implemented.
0b0001	Adds the BX instruction, and the T bit in the PSR.
0b0010	As for 0b0001, and adds the BLX instruction. PC loads have BX-like behavior.
0b0011	As for 0b0010, and guarantees that data-processing instructions in the A32 instruction set with the PC as the destination and the S bit clear have BX-like behavior.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Access to this field is **RO**.

Immediate, bits [23:20]

Indicates support for data-processing instructions with long immediates:

- The MOVT instruction
- The MOV instruction encodings with zero-extended 16-bit immediates.
- The T32 ADD and SUB instruction encodings with zero-extended 12-bit immediates, and the other ADD, ADR, and SUB encodings cross-referenced by the pseudocode for those encodings.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Immediate	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

IfThen, bits [19:16]

Indicates the implemented If-Then instructions in the T32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IfThen	Meaning
0b0000	None implemented.
0b0001	Adds the IT instructions, and the IT bits in the PSRs.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Extend, bits [15:12]

Indicates the implemented Extend instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Extend	Meaning
0b0000	No scalar sign-extend or zero-extend instructions are implemented, where scalar instructions means non-Advanced SIMD instructions.
0b0001	Adds the SXTB, SXTH, UXTB, and UXTH instructions.
0b0010	As for 0b0001, and adds the SXTB16, SXTAB, SXTAB16, SXTAH, UXTB16, UXTAB, UXTAB16, and UXTAH instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

Except_AR, bits [11:8]

Indicates the implemented A and R-profile exception-handling instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Except_AR	Meaning
0b0000	None implemented.
0b0001	Adds the SRS and RFE instructions, and the A and R-profile forms of the CPS instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Except, bits [7:4]

Indicates the implemented exception-handling instructions in the A32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Except	Meaning
0b0000	Not implemented. This indicates that the User bank and Exception return forms of the LDM and STM instructions are not implemented.
0b0001	Adds the LDM (exception return), LDM (user registers), and STM (user registers) instruction versions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Endian, bits [3:0]

Indicates the implemented Endian instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Endian	Meaning
0b0000	None implemented.
0b0001	Adds the SETEND instruction, and the E bit in the PSRs.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

Accessing ID_ISAR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_ISAR1;
elsif PSTATE.EL == EL2 then
    R[t] = ID_ISAR1;
elsif PSTATE.EL == EL3 then
    R[t] = ID_ISAR1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR2, Instruction Set Attribute Register 2

The ID_ISAR2 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR3, ID_ISAR4, and ID_ISAR5.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

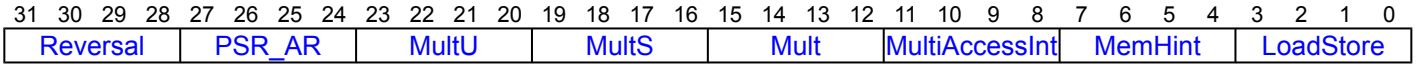
AArch32 System register ID_ISAR2 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR2_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_ISAR2 are UNDEFINED.

Attributes

ID_ISAR2 is a 32-bit register.

Field descriptions



Reversal, bits [31:28]

Indicates the implemented Reversal instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Reversal	Meaning
0b0000	None implemented.
0b0001	Adds the REV, REV16, and REVSH instructions.
0b0010	As for 0b0001, and adds the RBIT instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

PSR_AR, bits [27:24]

Indicates the implemented A and R-profile instructions to manipulate the PSR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PSR_AR	Meaning
0b0000	None implemented.
0b0001	Adds the MRS and MSR instructions, and the exception return forms of data-processing instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

The exception return forms of the data-processing instructions are:

- In the A32 instruction set, data-processing instructions with the PC as the destination and the S bit set. These instructions might be affected by the WithShifts attribute.
- In the T32 instruction set, the `SUBS PC, LR, #N` instruction.

Access to this field is **RO**.

MultU, bits [23:20]

Indicates the implemented advanced unsigned Multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MultU	Meaning
`0b0000`	None implemented.
`0b0001`	Adds the `UMULL` and `UMLAL` instructions.
`0b0010`	As for `0b0001`, and adds the `UMAAL` instruction.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0010`.

Access to this field is **RO**.

MultS, bits [19:16]

Indicates the implemented advanced signed Multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MultS	Meaning
`0b0000`	None implemented.
`0b0001`	Adds the `SMULL` and `SMLAL` instructions.
`0b0010`	As for `0b0001`, and adds the `SMLABB`, `SMLABT`, `SMLALBB`, `SMLALBT`, `SMLALTB`, `SMLALTT`, `SMLATB`, `SMLATT`, `SMLAWB`, `SMLAWT`, `SMULBB`, `SMULBT`, `SMULTB`, `SMULTT`, `SMULWB`, and `SMULWT` instructions. Also adds the Q bit in the PSRs.
`0b0011`	As for `0b0010`, and adds the `SMLAD`, `SMLADX`, `SMLALD`, `SMLALDX`, `SMLSD`, `SMLSDX`, `SMLSLD`, `SMLSLDX`, `SMMLA`, `SMMLAR`, `SMMLS`, `SMMLSR`, `SMMUL`, `SMMULR`, `SMUAD`, `SMUADX`, `SMUSD`, and `SMUSDX` instructions.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0011`.

Access to this field is **RO**.

Mult, bits [15:12]

Indicates the implemented additional Multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Mult	Meaning
`0b0000`	No additional instructions implemented. This means only `MUL` is implemented.
`0b0001`	Adds the `MLA` instruction.
`0b0010`	As for `0b0001`, and adds the `MLS` instruction.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0010`.

Access to this field is **RO**.

MultiAccessInt, bits [11:8]

Indicates the support for interruptible multi-access instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MultiAccessInt	Meaning
0b0000	No support. This means the LDM and STM instructions are not interruptible.
0b0001	LDM and STM instructions are restartable.
0b0010	LDM and STM instructions are continuable.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

MemHint, bits [7:4]

Indicates the implemented Memory Hint instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MemHint	Meaning
0b0000	None implemented.
0b0001	Adds the PLD instruction.
0b0010	Adds the PLD instruction. (0b0001 and 0b0010 have identical effects.)
0b0011	As for 0b0001 (or 0b0010), and adds the PLI instruction.
0b0100	As for 0b0011, and adds the PLDW instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0100.

Access to this field is **RO**.

LoadStore, bits [3:0]

Indicates the implemented additional load/store instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LoadStore	Meaning
0b0000	No additional load/store instructions implemented.
0b0001	Adds the LDRD and STRD instructions.
0b0010	As for 0b0001, and adds the Load Acquire (LDAB, LDAH, LDA, LDAEXB, LDAEXH, LDAEX, LDAEXD) and Store Release (STLB, STLH, STL, STLEXB, STLEXH, STLEX, STLEXD) instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

Accessing ID_ISAR2

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_ISAR2;
elsif PSTATE.EL == EL2 then
    R[t] = ID_ISAR2;
elsif PSTATE.EL == EL3 then
    R[t] = ID_ISAR2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR3, Instruction Set Attribute Register 3

The ID_ISAR3 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR4, and ID_ISAR5.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

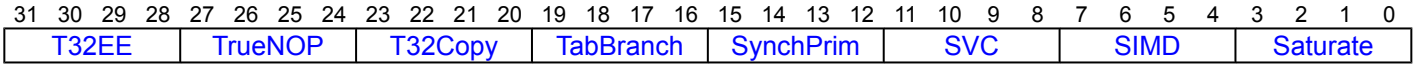
AArch32 System register ID_ISAR3 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR3_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_ISAR3 are UNDEFINED.

Attributes

ID_ISAR3 is a 32-bit register.

Field descriptions



T32EE, bits [31:28]

Indicates the implemented T32EE instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

T32EE	Meaning
0b0000	None implemented.
0b0001	Adds the ENTERX and LEAVEX instructions, and modifies the load behavior to include null checking.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

TrueNOP, bits [27:24]

Indicates the implemented true NOP instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TrueNOP	Meaning
0b0000	None implemented. This means there are no NOP instructions that do not have any register dependencies.
0b0001	Adds true NOP instructions in both the T32 and A32 instruction sets. This also permits additional NOP-compatible hints.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

T32Copy, bits [23:20]

Indicates the support for T32 non flag-setting `MOV` instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

T32Copy	Meaning
`0b0000`	Not supported. This means that in the T32 instruction set, encoding T1 of the `MOV` (register) instruction does not support a copy from a low register to a low register.
`0b0001`	Adds support for T32 instruction set encoding T1 of the `MOV` (register) instruction, copying from a low register to a low register.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0001`.

Access to this field is **RO**.

TabBranch, bits [19:16]

Indicates the implemented Table Branch instructions in the T32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TabBranch	Meaning
`0b0000`	None implemented.
`0b0001`	Adds the `TBB` and `TBH` instructions.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0001`.

Access to this field is **RO**.

SynchPrim, bits [15:12]

Used in conjunction with ID_ISAR4.SynchPrim_frac to indicate the implemented Synchronization Primitive instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SynchPrim	Meaning
`0b0000`	If SynchPrim_frac == `0b000`, no Synchronization Primitives implemented.
`0b0001`	If SynchPrim_frac == `0b000`, adds the `LDREX` and `STREX` instructions. If SynchPrim_frac == `0b011`, also adds the `CLREX`, `LDREXB`, `STREXB`, and `STREXH` instructions.
`0b0010`	If SynchPrim_frac == `0b000`, as for [`0b001`, `0b011`] and also adds the `LDREXD` and `STREXD` instructions.

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is `0b0010`.

Access to this field is **RO**.

SVC, bits [11:8]

Indicates the implemented `SVC` instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SVC	Meaning
0b0000	Not implemented.
0b0001	Adds the SVC instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

SIMD, bits [7:4]

Indicates the implemented SIMD instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMD	Meaning
0b0000	None implemented.
0b0001	Adds the SSAT and USAT instructions, and the Q bit in the PSRs.
0b0011	As for 0b0001, and adds the PKHBT, PKHTB, QADD16, QADD8, QASX, QSUB16, QSUB8, QSAX, SADD16, SADD8, SASX, SEL, SHADD16, SHADD8, SHASX, SHSUB16, SHSUB8, SHSAX, SSAT16, SSUB16, SSUB8, SSAX, SXTAB16, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSUB16, UHSUB8, UHSAX, UQADD16, UQADD8, UQASX, UQSUB16, UQSUB8, UQSAX, USAD8, USADA8, USAT16, USUB16, USUB8, USAX, UXTAB16, and UXTB16 instructions. Also adds support for the GE[3:0] bits in the PSRs.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

The SIMD field relates only to implemented instructions that perform SIMD operations on the general-purpose registers. In an implementation that supports Advanced SIMD and floating-point instructions, MVFR0, MVFR1, and MVFR2 give information about the implemented Advanced SIMD instructions.

Access to this field is **RO**.

Saturate, bits [3:0]

Indicates support for Saturate instructions QADD, QDADD, QDSUB, and QSUB instructions, and the Q bit in the PSRs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Saturate	Meaning
0b0000	None implemented. This means no non-Advanced SIMD saturate instructions are implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Accessing ID_ISAR3

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_ISAR3;
elsif PSTATE.EL == EL2 then
    R[t] = ID_ISAR3;
elsif PSTATE.EL == EL3 then
    R[t] = ID_ISAR3;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR4, Instruction Set Attribute Register 4

The ID_ISAR4 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, and ID_ISAR5.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

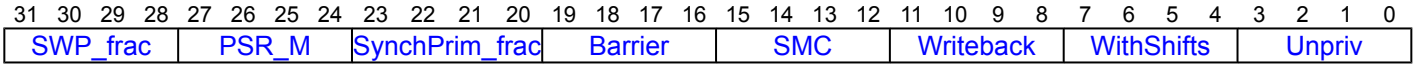
AArch32 System register ID_ISAR4 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR4_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_ISAR4 are UNDEFINED.

Attributes

ID_ISAR4 is a 32-bit register.

Field descriptions



SWP_frac, bits [31:28]

Indicates support for the memory system locking the bus for `SWP` or `SWPB` instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SWP_frac	Meaning
`0b0000`	`SWP` or `SWPB` instructions not implemented.
`0b0001`	`SWP` or `SWPB` implemented but only in a uniprocessor context. `SWP` and `SWPB` do not guarantee whether memory accesses from other Requesters can come between the load memory access and the store memory access of the `SWP` or `SWPB`.

All other values are reserved. This field is valid only if ID_ISAR0.Swap is `0b0000`.

In Armv8-A, the only permitted value is `0b0000`.

Access to this field is **RO**.

PSR_M, bits [27:24]

Indicates the implemented M-profile instructions to modify the PSRs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PSR_M	Meaning
`0b0000`	None implemented.
`0b0001`	Adds the M-profile forms of the `CPS`, `MRS`, and `MSR` instructions.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0000`.

Access to this field is **RO**.

SynchPrim_frac, bits [23:20]

Used in conjunction with ID_ISAR3.SynchPrim to indicate the implemented Synchronization Primitive instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SynchPrim_frac	Meaning
0b0000	If SynchPrim == 0b0000, no Synchronization Primitives implemented. If SynchPrim == 0b0001, adds the LDREX and STREX instructions. If SynchPrim == 0b0010, also adds the CLREX, LDREXB, LDREXH, STREXB, STREXH, LDREXD, and STREXD instructions.
0b0011	If SynchPrim == 0b0001, adds the LDREX, STREX, CLREX, LDREXB, LDREXH, STREXB, and STREXH instructions.

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

Barrier, bits [19:16]

Indicates support for Barrier instructions DMB, DSB, and ISB in the T32 and A32 instruction sets.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Barrier	Meaning
0b0000	None implemented. Barrier operations are provided only as System instructions in the (coproc==0b1111) encoding space.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

SMC, bits [15:12]

Indicates the implemented SMC instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SMC	Meaning
0b0000	None implemented.
0b0001	Adds the SMC instruction.

All other values are reserved.

In Armv8-A, the permitted values are:

- If EL3 is implemented, the only permitted value is 0b0001.
- If neither EL3 nor EL2 is implemented, the only permitted value is 0b0000.

Access to this field is **RO**.

Writeback, bits [11:8]

Indicates the support for Writeback addressing modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Writeback	Meaning
0b0000	Basic support. Only the LDM, STM, PUSH, POP, SRS, and RFE instructions support writeback addressing modes. These instructions support all of their writeback addressing modes.
0b0001	Adds support for all of the writeback addressing modes.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

WithShifts, bits [7:4]

Indicates the support for instructions with shifts.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WithShifts	Meaning
0b0000	Nonzero shifts supported only in MOV and shift instructions.
0b0001	Adds support for shifts of loads and stores over the range LSL 0-3.
0b0011	As for 0b0001, and adds support for other constant shift options, both on load/store and other instructions.
0b0100	As for 0b0011, and adds support for register-controlled shift options.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0100.

Access to this field is **RO**.

Unpriv, bits [3:0]

Indicates the implemented unprivileged instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Unpriv	Meaning
0b0000	None implemented. No T variant instructions are implemented.
0b0001	Adds the LDRBT, LDRT, STRBT, and STRT instructions.
0b0010	As for 0b0001, and adds the LDRHT, LDRSBT, LDRSHT, and STRHT instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

Accessing ID_ISAR4

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0010	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_ISAR4;
elsif PSTATE.EL == EL2 then
    R[t] = ID_ISAR4;
elsif PSTATE.EL == EL3 then
    R[t] = ID_ISAR4;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR5, Instruction Set Attribute Register 5

The ID_ISAR5 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, and ID_ISAR4.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

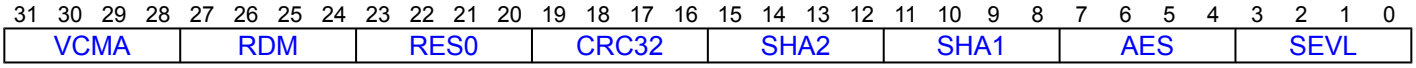
AArch32 System register ID_ISAR5 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR5_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_ISAR5 are UNDEFINED.

Attributes

ID_ISAR5 is a 32-bit register.

Field descriptions



VCMA, bits [31:28]

Indicates AArch32 support for complex number addition and multiplication where numbers are stored in vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VCMA	Meaning
0b0000	The VCMLA and VCADD instructions are not implemented in AArch32.
0b0001	The VCMLA and VCADD instructions are implemented in AArch32.

All other values are reserved.

FEAT_FCMA implements the functionality identified by 0b0001.

From Armv8.3, the value 0b0000 is not permitted.

Access to this field is **RO**.

RDM, bits [27:24]

Indicates support for the VQRDMLAH and VQRDMLSH instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RDM	Meaning
0b0000	No VQRDMLAH and VQRDMLSH instructions implemented.
0b0001	VQRDMLAH and VQRDMLSH instructions implemented.

All other values are reserved.

FEAT_RDM implements the functionality identified by the value 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

CRC32, bits [19:16]

Indicates support for the CRC32 instructions CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, and CRC32CW in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CRC32	Meaning
0b0000	CRC32 instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_CRC32 implements the functionality identified by the value 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is **RO**.

SHA2, bits [15:12]

Indicates support for the SHA2 instructions SHA256H, SHA256H2, SHA256SU0, and SHA256SU1 in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA2	Meaning
0b0000	No SHA2 instructions implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

SHA1, bits [11:8]

Indicates support for the SHA1 instructions SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA1	Meaning
0b0000	No SHA1 instructions implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

AES, bits [7:4]

Indicates support for the AES instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AES	Meaning
0b0000	No AES instructions implemented.
0b0001	AESE, AESD, AESMC, and AESIMC implemented.
0b0010	As for 0b0001, plus VMULL (polynomial) instructions operating on 64-bit data quantities.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

Access to this field is **RO**.

SEVL, bits [3:0]

Indicates support for the SEVL instruction in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEVL	Meaning
0b0000	SEVL is implemented as a NOP.
0b0001	SEVL is implemented as Send Event Local.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Accessing ID_ISAR5

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0010	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_ISAR5;
elsif PSTATE.EL == EL2 then
    R[t] = ID_ISAR5;
elsif PSTATE.EL == EL3 then
    R[t] = ID_ISAR5;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR6, Instruction Set Attribute Register 6

The ID_ISAR6 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, ID_ISAR4, and ID_ISAR5.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_ISAR6 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR6_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_ISAR6 are UNDEFINED.

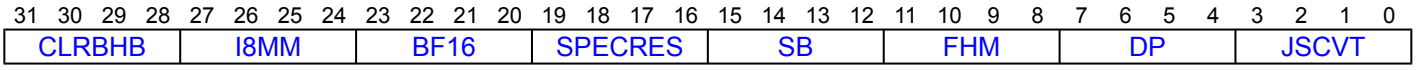
Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_ISAR6 is a 32-bit register.

Field descriptions



CLRBHB, bits [31:28]

Indicates support for the CLRBHB instruction in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CLRBHB	Meaning
0b0000	CLRBHB instruction is not implemented.
0b0001	CLRBHB instruction is implemented.

All other values are reserved.

FEAT_CLRBHB implements the functionality identified by 0b0001.

From Armv8.9, the value 0b0000 is not permitted.

Access to this field is **RO**.

I8MM, bits [27:24]

Indicates support for Advanced SIMD and floating-point Int8 matrix multiplication instructions VSMMLA, VSUDOT, VUMMLA, VUSMMLA, and VUSDOT in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I8MM	Meaning
0b0000	Int8 matrix multiplication instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_AA32I8MM implements the functionality identified by 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

BF16, bits [23:20]

Indicates support for Advanced SIMD and floating-point BFloat16 instructions VCVT, VCVTB, VCVTT, VDOT, VFMAB, VFMAT, and VMMLA instructions with BF16 operand or result types in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BF16	Meaning
0b0000	BFloat16 instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_AA32BF16 implements the functionality identified by 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

SPECRES, bits [19:16]

Indicates support for prediction invalidation instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPECRES	Meaning
0b0000	Prediction invalidation instructions are not implemented.
0b0001	CFPRCTX, DVPRCTX, and CPPRCTX instructions are implemented.
0b0010	As 0b0001, and the COSPRCTX instruction is implemented.

All other values are reserved.

FEAT_SPECRES implements the functionality identified by 0b0001.

FEAT_SPECRES2 implements the functionality identified by 0b0010.

From Armv8.5, the value 0b0000 is not permitted.

From Armv8.9, the value 0b0001 is not permitted.

Access to this field is **RO**.

SB, bits [15:12]

Indicates support for SB instruction in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SB	Meaning
0b0000	SB instruction is not implemented.
0b0001	SB instruction is implemented.

All other values are reserved.

From Armv8.5, the value 0b0000 is not permitted.

Access to this field is **RO**.

FHM, bits [11:8]

Indicates support for Advanced SIMD and floating-point `VFMAL` and `VMFSL` instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FHM	Meaning
`0b0000`	`VFMAL` and `VMFSL` instructions not implemented.
`0b0001`	`VFMAL` and `VMFSL` instructions implemented.

FEAT_FHM implements the functionality identified by the value `0b0001`.

Access to this field is **RO**.

DP, bits [7:4]

Indicates support for dot product instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DP	Meaning
`0b0000`	No dot product instructions implemented.
`0b0001`	`VUDOT` and `VSDOT` instructions implemented.

All other values are reserved.

FEAT_DotProd implements the functionality identified by the value `0b0001`.

Access to this field is **RO**.

JSCVT, bits [3:0]

Indicates support for the Javascript conversion instruction in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

JSCVT	Meaning
`0b0000`	The `VJCVT` instruction is not implemented.
`0b0001`	The `VJCVT` instruction is implemented.

All other values are reserved.

In Armv8.0, the only permitted value is `0b0000`.

FEAT_JSCVT implements the functionality identified by `0b0001`.

From Armv8.3, if Advanced SIMD or Floating-point is implemented, the value `0b0000` is not permitted.

From Armv8.3, if Advanced SIMD or Floating-point is not implemented, the only permitted value is `0b0000`.

Access to this field is **RO**.

Accessing ID_ISAR6

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0010	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
(IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_ISAR6) || boolean IMPLEMENTATION_DEFINED "ID_ISAR6
trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
(IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_ISAR6) || boolean IMPLEMENTATION_DEFINED "ID_ISAR6
trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_ISAR6;
elsif PSTATE.EL == EL2 then
    R[t] = ID_ISAR6;
elsif PSTATE.EL == EL3 then
    R[t] = ID_ISAR6;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR0, Memory Model Feature Register 0

The ID_MMFR0 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

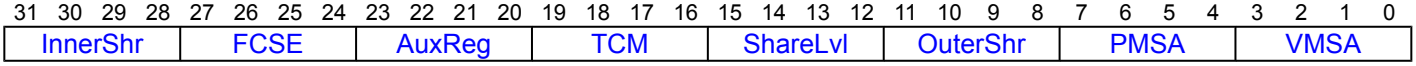
AArch32 System register ID_MMFR0 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_MMFR0 are UNDEFINED.

Attributes

ID_MMFR0 is a 32-bit register.

Field descriptions



InnerShr, bits [31:28]

Innermost Shareability. Indicates the innermost shareability domain implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

InnerShr	Meaning
0b0000	Implemented as Non-cacheable.
0b0001	Implemented with hardware coherency support.
0b1111	Shareability ignored.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000, 0b0001, and 0b1111.

This field is valid only if the implementation supports two levels of shareability, as indicated by ID_MMFR0.ShareLvl having the value 0b0001.

When ID_MMFR0.ShareLvl is zero, this field is UNKNOWN.

Access to this field is **RO**.

FCSE, bits [27:24]

Indicates whether the implementation includes the FCSE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FCSE	Meaning
0b0000	Not supported.
0b0001	Support for FCSE.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

AuxReg, bits [23:20]

Auxiliary Registers. Indicates support for Auxiliary registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AuxReg	Meaning
0b0000	None supported.
0b0001	Support for Auxiliary Control Register only.
0b0010	Support for Auxiliary Fault Status Registers (AIFSR and ADFSR) and Auxiliary Control Register.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Note

Accesses to unimplemented Auxiliary registers are UNDEFINED.

Access to this field is **RO**.

TCM, bits [19:16]

Indicates support for TCMs and associated DMAs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TCM	Meaning
0b0000	Not supported.
0b0001	Support is IMPLEMENTATION DEFINED.
0b0010	Support for TCM only, Armv6 implementation.
0b0011	Support for TCM and DMA, Armv6 implementation.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

ShareLvl, bits [15:12]

Shareability Levels. Indicates the number of shareability levels implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ShareLvl	Meaning
0b0000	One level of shareability implemented.
0b0001	Two levels of shareability implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

OuterShr, bits [11:8]

Outermost Shareability. Indicates the outermost shareability domain implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OuterShr	Meaning
0b0000	Implemented as Non-cacheable.
0b0001	Implemented with hardware coherency support.
0b1111	Shareability ignored.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000, 0b0001, and 0b1111.

Access to this field is **RO**.

PMSA, bits [7:4]

Indicates support for a PMSA.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMSA	Meaning
0b0000	Not supported.
0b0001	Support for IMPLEMENTATION DEFINED PMSA.
0b0010	Support for PMSAv6, with a Cache Type Register implemented.
0b0011	Support for PMSAv7, with support for memory subsections. Armv7-R profile.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

VMSA, bits [3:0]

Indicates support for a VMSA.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMSA	Meaning
0b0000	Not supported.
0b0001	Support for IMPLEMENTATION DEFINED VMSA.
0b0010	Support for VMSAv6, with Cache and TLB Type Registers implemented.
0b0011	Support for VMSAv7, with support for remapping and the Access flag. ARMv7-A profile.
0b0100	As for 0b0011, and adds support for the PXN bit in the Short-descriptor translation table format descriptors.
0b0101	As for 0b0100, and adds support for the Long-descriptor translation table format.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0101.

Access to this field is **RO**.

Accessing ID_MMFR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0001	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_MMFR0;
elsif PSTATE.EL == EL2 then
    R[t] = ID_MMFR0;
elsif PSTATE.EL == EL3 then
    R[t] = ID_MMFR0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR1, Memory Model Feature Register 1

The ID_MMFR1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_MMFR1 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_MMFR1 are UNDEFINED.

Attributes

ID_MMFR1 is a 32-bit register.

Field descriptions

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
BPred	L1TstCln	L1Uni	L1Hvd	L1UniSW	L1HvdSW	L1UniVA	L1HvdVA

BPred, bits [31:28]

Branch Predictor. Indicates branch predictor management requirements.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BPred	Meaning
0b0000	No branch predictor, or no MMU present. Implies a fixed MPU configuration.
0b0001	Branch predictor requires flushing on: • Enabling or disabling a stage of address translation. • Writing new data to instruction locations. • Writing new mappings to the translation tables. • Changes to the TTBR0, TTBR1, or TTBCR registers. • Changes to the ContextID or ASID, or to the FCSE ProcessID if this is supported.
0b0010	Branch predictor requires flushing on: • Enabling or disabling a stage of address translation. • Writing new data to instruction locations. • Writing new mappings to the translation tables. • Any change to the TTBR0, TTBR1, or TTBCR registers without a change to the corresponding ContextID or ASID, or FCSE ProcessID if this is supported.
0b0011	Branch predictor requires flushing only on writing new data to instruction locations.
0b0100	For execution correctness, branch predictor requires no flushing at any time.

All other values are reserved.

In Armv8-A, the permitted values are 0b0010, 0b0011, or 0b0100. For values other than 0b0000 and 0b0100, the Arm Architecture Reference Manual, or the product documentation, might give more information about the required maintenance.

Access to this field is **RO**.

L1TstCln, bits [27:24]

Level 1 cache Test and Clean. Indicates the supported Level 1 data cache test and clean operations, for Harvard or unified cache implementations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1TstCln	Meaning
0b0000	None supported.
0b0001	Supported Level 1 data cache test and clean operations are: • Test and clean data cache.
0b0010	As for 0b0001, and adds: • Test, clean, and invalidate data cache.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1Uni, bits [23:20]

Level 1 Unified cache. Indicates the supported entire Level 1 cache maintenance operations for a unified cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1Uni	Meaning
0b0000	None supported.
0b0001	Supported entire Level 1 cache operations are: • Invalidate cache, including branch predictor if appropriate. • Invalidate branch predictor, if appropriate.
0b0010	As for 0b0001, and adds: • Clean cache, using a recursive model that uses the cache dirty status bit. • Clean and invalidate cache, using a recursive model that uses the cache dirty status bit.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1Hvd, bits [19:16]

Level 1 Harvard cache. Indicates the supported entire Level 1 cache maintenance operations for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1Hvd	Meaning
0b0000	None supported.
0b0001	Supported entire Level 1 cache operations are: • Invalidate instruction cache, including branch predictor if appropriate. • Invalidate branch predictor, if appropriate.
0b0010	As for 0b0001, and adds: • Invalidate data cache. • Invalidate data cache and instruction cache, including branch predictor if appropriate.
0b0011	As for 0b0010, and adds: • Clean data cache, using a recursive model that uses the cache dirty status bit. • Clean and invalidate data cache, using a recursive model that uses the cache dirty status bit.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1UniSW, bits [15:12]

Level 1 Unified cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a unified cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1UniSW	Meaning
0b0000	None supported.
0b0001	Supported Level 1 unified cache line maintenance operations by set/way are: • Clean cache line by set/way.
0b0010	As for 0b0001, and adds: • Clean and invalidate cache line by set/way.
0b0011	As for 0b0010, and adds: • Invalidate cache line by set/way.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1HvdSW, bits [11:8]

Level 1 Harvard cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdSW	Meaning
0b0000	None supported.
0b0001	Supported Level 1 Harvard cache line maintenance operations by set/way are: • Clean data cache line by set/way. • Clean and invalidate data cache line by set/way.
0b0010	As for 0b0001, and adds: • Invalidate data cache line by set/way.
0b0011	As for 0b0010, and adds: • Invalidate instruction cache line by set/way.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1UniVA, bits [7:4]

Level 1 Unified cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a unified cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1UniVA	Meaning
0b0000	None supported.
0b0001	Supported Level 1 unified cache line maintenance operations by VA are: • Clean cache line by VA. • Invalidate cache line by VA. • Clean and invalidate cache line by VA.
0b0010	As for 0b0001, and adds: • Invalidate branch predictor by VA, if branch predictor is implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1HvdVA, bits [3:0]

Level 1 Harvard cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdVA	Meaning
0b0000	None supported.
0b0001	Supported Level 1 Harvard cache line maintenance operations by VA are: • Clean data cache line by VA. • Invalidate data cache line by VA. • Clean and invalidate data cache line by VA. • Clean instruction cache line by VA.
0b0010	As for 0b0001, and adds: • Invalidate branch predictor by VA, if branch predictor is implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

Accessing ID_MMFR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0001	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_MMFR1;
elsif PSTATE.EL == EL2 then
    R[t] = ID_MMFR1;
elsif PSTATE.EL == EL3 then
    R[t] = ID_MMFR1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR2, Memory Model Feature Register 2

The ID_MMFR2 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

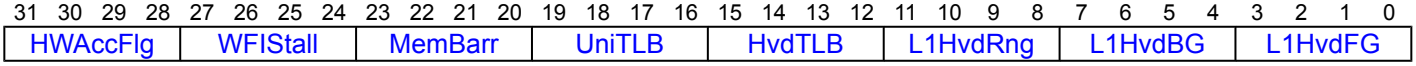
AArch32 System register ID_MMFR2 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR2_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_MMFR2 are UNDEFINED.

Attributes

ID_MMFR2 is a 32-bit register.

Field descriptions



HWAccFlg, bits [31:28]

Hardware Access Flag. In earlier versions of the Arm Architecture, this field indicates support for a Hardware Access flag, as part of the VMSAv7 implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HWAccFlg	Meaning
0b0000	Not supported.
0b0001	Support for VMSAv7 Access flag, updated in hardware.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

WFIStall, bits [27:24]

Wait For Interrupt Stall. Indicates the support for Wait For Interrupt (WFI) stalling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WFIStall	Meaning
0b0000	Not supported.
0b0001	Support for WFI stalling.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

MemBarr, bits [23:20]

Memory Barrier. Indicates the supported memory barrier System instructions in the (coproc == 1111) encoding space.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MemBarr	Meaning
0b0000	None supported.
0b0001	Supported memory barrier System instructions are: • Data Synchronization Barrier (DSB).
0b0010	As for 0b0001, and adds: • Instruction Synchronization Barrier (ISB). • Data Memory Barrier (DMB).

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Arm deprecates the use of these operations. ID_ISAR4.Barrier_instrs indicates the level of support for the preferred barrier instructions.

Access to this field is **RO**.

UniTLB, bits [19:16]

Unified TLB. Indicates the supported TLB maintenance operations, for a unified TLB implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UniTLB	Meaning
0b0000	Not supported.
0b0001	Supported unified TLB maintenance operations are: • Invalidate all entries in the TLB. • Invalidate TLB entry by VA.
0b0010	As for 0b0001, and adds: • Invalidate TLB entries by ASID match.
0b0011	As for 0b0010, and adds: • Invalidate instruction TLB and data TLB entries by VA All ASID. This is a shared unified TLB operation
0b0100	As for 0b0011, and adds: • Invalidate Hyp mode unified TLB entry by VA. • Invalidate entire Non-secure PL1&0 unified TLB. • Invalidate entire Hyp mode unified TLB.
0b0101	As for 0b0100, and adds the following operations: TLBIMVALIS, TLBIMVAALIS, TLBIMVALHIS, TLBIMVAL, TLBIMVAAL,TLBIMVALH.
0b0110	As for 0b0101, and adds the following operations: TLBIIPAS2IS, TLBIIPAS2LIS, TLBIIPAS2, TLBIIPAS2L.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0110.

Access to this field is **RO**.

HvdTLB, bits [15:12]

If the value of ID_MMFR2.UniTLB is not 0b0000, then the meaning of this field is IMPLEMENTATION DEFINED. Arm deprecates the use of this field by software.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

L1HvdRng, bits [11:8]

Level 1 Harvard cache Range. Indicates the supported Level 1 cache maintenance range operations, for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdRng	Meaning
0b0000	Not supported.
0b0001	Supported Level 1 Harvard cache maintenance range operations are: • Invalidate data cache range by VA. • Invalidate instruction cache range by VA. • Clean data cache range by VA. • Clean and invalidate data cache range by VA.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1HvdBG, bits [7:4]

Level 1 Harvard cache Background fetch. Indicates the supported Level 1 cache background fetch operations, for a Harvard cache implementation. When supported, background fetch operations are non-blocking operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdBG	Meaning
0b0000	Not supported.
0b0001	Supported Level 1 Harvard cache background fetch operations are: • Fetch instruction cache range by VA. • Fetch data cache range by VA.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

L1HvdFG, bits [3:0]

Level 1 Harvard cache Foreground fetch. Indicates the supported Level 1 cache foreground fetch operations, for a Harvard cache implementation. When supported, foreground fetch operations are blocking operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdFG	Meaning
0b0000	Not supported.
0b0001	Supported Level 1 Harvard cache foreground fetch operations are: • Fetch instruction cache range by VA. • Fetch data cache range by VA.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

Accessing ID_MMFR2

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0001	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_MMFR2;
elsif PSTATE.EL == EL2 then
    R[t] = ID_MMFR2;
elsif PSTATE.EL == EL3 then
    R[t] = ID_MMFR2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR3, Memory Model Feature Register 3

The ID_MMFR3 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

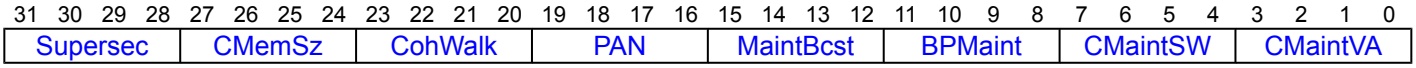
AArch32 System register ID_MMFR3 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR3_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_MMFR3 are UNDEFINED.

Attributes

ID_MMFR3 is a 32-bit register.

Field descriptions



Supersec, bits [31:28]

Supersections. On a VMSA implementation, indicates whether Supersections are supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Supersec	Meaning
0b0000	Supersections supported.
0b1111	Supersections not supported.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b1111.

Access to this field is **RO**.

CMemSz, bits [27:24]

Cached Memory Size. Indicates the physical memory size supported by the caches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CMemSz	Meaning
0b0000	4GB, corresponding to a 32-bit physical address range.
0b0001	64GB, corresponding to a 36-bit physical address range.
0b0010	1TB or more, corresponding to a 40-bit or larger physical address range.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000, 0b0001, and 0b0010.

Access to this field is **RO**.

CohWalk, bits [23:20]

Coherent Walk. Indicates whether Translation table updates require a clean to the Point of Unification.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CohWalk	Meaning
0b0000	Updates to the translation tables require a clean to the Point of Unification to ensure visibility by subsequent translation table walks.
0b0001	Updates to the translation tables do not require a clean to the Point of Unification to ensure visibility by subsequent translation table walks.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

PAN, bits [19:16]

Privileged Access Never. Indicates support for the PAN bit in CPSR, SPSR, and DSPSR in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PAN	Meaning
0b0000	PAN not supported.
0b0001	PAN supported.
0b0010	PAN supported and ATS1CPRP and ATS1CPWP instructions supported.

All other values are reserved.

FEAT_PAN implements the functionality identified by the value 0b0001.

FEAT_PAN2 implements the functionality added by the value 0b0010.

From Armv8.1, the value 0b0000 is not permitted.

From Armv8.2, the value 0b0001 is not permitted.

Access to this field is **RO**.

MaintBcst, bits [15:12]

Maintenance Broadcast. Indicates whether Cache, TLB, and branch predictor operations are broadcast.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MaintBcst	Meaning
0b0000	Cache, TLB, and branch predictor operations only affect local structures.
0b0001	Cache and branch predictor operations affect structures according to shareability and defined behavior of instructions. TLB operations only affect local structures.
0b0010	Cache, TLB, and branch predictor operations affect structures according to shareability and defined behavior of instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

BPMaint, bits [11:8]

Branch Predictor Maintenance. Indicates the supported branch predictor maintenance operations in an implementation with hierarchical cache maintenance operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BPMaint	Meaning
0b0000	None supported.
0b0001	Supported branch predictor maintenance operations are: • Invalidate all branch predictors.
0b0010	As for 0b0001, and adds: • Invalidate branch predictors by VA.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is **RO**.

CMaintSW, bits [7:4]

Cache Maintenance by Set/Way. Indicates the supported cache maintenance operations by set/way, in an implementation with hierarchical caches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CMaintSW	Meaning
0b0000	None supported.
0b0001	Supported hierarchical cache maintenance instructions by set/way are: • Invalidate data cache by set/way. • Clean data cache by set/way. • Clean and invalidate data cache by set/way.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

In a unified cache implementation, the data cache maintenance operations apply to the unified caches.

Access to this field is **RO**.

CMaintVA, bits [3:0]

Cache Maintenance by Virtual Address. Indicates the supported cache maintenance operations by VA, in an implementation with hierarchical caches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CMaintVA	Meaning
0b0000	None supported.
0b0001	Supported hierarchical cache maintenance operations by VA are: • Invalidate data cache by VA. • Clean data cache by VA. • Clean and invalidate data cache by VA. • Invalidate instruction cache by VA. • Invalidate all instruction cache entries.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

In a unified cache implementation, data cache maintenance operations apply to the unified caches, and the instruction cache maintenance instructions are not implemented.

Access to this field is **RO**.

Accessing ID_MMFR3

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0001	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_MMFR3;
elsif PSTATE.EL == EL2 then
    R[t] = ID_MMFR3;
elsif PSTATE.EL == EL3 then
    R[t] = ID_MMFR3;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR4, Memory Model Feature Register 4

The ID_MMFR4 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

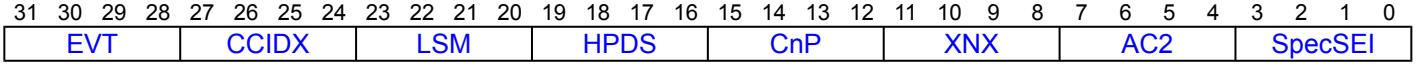
AArch32 System register ID_MMFR4 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR4_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_MMFR4 are UNDEFINED.

Attributes

ID_MMFR4 is a 32-bit register.

Field descriptions



EVT, bits [31:28]

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the HCR2.{TTLBIS, TOCU, TICAB, TID4} traps.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EVT	Meaning
0b0000	HCR2.{TTLBIS, TOCU, TICAB, TID4} traps are not supported.
0b0001	HCR2.{TOCU, TICAB, TID4} traps are supported. HCR2.TTLBIS trap is not supported.
0b0010	HCR2.{TTLBIS, TOCU, TICAB, TID4} traps are supported.

All other values are reserved.

FEAT_EVT implements the functionality identified by the values 0b0001 and 0b0010.

If EL2 is not implemented or does not support AArch32, the only permitted value is 0b0000.

From Armv8.5, if EL2 is implemented and supports AArch32, the value 0b0001 is not permitted.

Access to this field is **RO**.

CCIDX, bits [27:24]

Support for use of the revised CCSIDR format and the presence of the CCSIDR2 is indicated.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCIDX	Meaning
0b0000	32-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is not implemented.
0b0001	64-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is implemented.

All other values are reserved.

FEAT_CCIDX implements the functionality identified by `0b0001`.

From Armv8.3, the permitted values are `0b0000` and `0b0001`.

Access to this field is **RO**.

LSM, bits [23:20]

Indicates support for LSMAOE and nTLSMD bits in HSCTLR and SCTLR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LSM	Meaning
`0b0000`	LSMAOE and nTLSMD bits not supported.
`0b0001`	LSMAOE and nTLSMD bits supported.

All other values are reserved.

FEAT_LSMAOC implements the functionality identified by the value `0b0001`.

From Armv8.2, the permitted values are `0b0000` and `0b0001`.

Access to this field is **RO**.

HPDS, bits [19:16]

Hierarchical permission disables bits in translation tables.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HPDS	Meaning
`0b0000`	Disabling of hierarchical controls not supported.
`0b0001`	Supports disabling of hierarchical controls using the TTBCR2.HPD0, TTBCR2.HPD1, and HTCR.HPD bits.
`0b0010`	As for value `0b0001`, and adds possible hardware allocation of bits[62:59] of the Translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use.

All other values are reserved.

FEAT_AA32HPD implements the functionality identified by the value `0b0001`.

FEAT_HPDS2 implements the functionality added by the value `0b0010`.

Note

The value `0b0000` implies that the encoding for TTBCR2 is UNDEFINED.

Access to this field is **RO**.

CnP, bits [15:12]

Common not Private translations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CnP	Meaning
`0b0000`	Common not Private translations not supported.
`0b0001`	Common not Private translations supported.

All other values are reserved.

FEAT_TTCNP implements the functionality identified by the value `0b0001`.

From Armv8.2, the value 0b0000 is not permitted.

Access to this field is **RO**.

XNX, bits [11:8]

Support for execute-never control distinction by Exception level at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

XNX	Meaning
0b0000	Distinction between EL0 and EL1 execute-never control at stage 2 not supported.
0b0001	Distinction between EL0 and EL1 execute-never control at stage 2 supported.

All other values are reserved.

FEAT_XNX implements the functionality identified by the value 0b0001.

When FEAT_XNX is implemented:

- If all of the following conditions are true, it is IMPLEMENTATION DEFINED whether the value of ID_MMFR4.XNX is 0b0000 or 0b0001:
  - ID_AA64MMFR1_EL1.XNX ==1.
  - EL2 cannot use AArch32.
  - EL1 can use AArch32.
- If EL2 can use AArch32 then the value 0b0000 is not permitted.

Access to this field is **RO**.

AC2, bits [7:4]

Indicates the extension of the ACTLR and HACTLR registers using ACTLR2 and HACTLR2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AC2	Meaning
0b0000	ACTLR2 and HACTLR2 are not implemented.
0b0001	ACTLR2 and HACTLR2 are implemented.

All other values are reserved.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.2, the only permitted value is 0b0001.

Access to this field is **RO**.

SpecSEI, bits [3:0]

When FEAT_RAS is implemented:

Describes whether the PE can generate SError exceptions from speculative reads of memory, including speculative instruction fetches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SpecSEI	Meaning
0b0000	The PE never generates an SError exception due to an External abort on a speculative read.
0b0001	The PE might generate an SError exception due to an External abort on a speculative read.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing ID_MMFR4

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
(IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_MMFR4) || boolean IMPLEMENTATION_DEFINED "ID_MMFR4
trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
(IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_MMFR4) || boolean IMPLEMENTATION_DEFINED "ID_MMFR4
trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_MMFR4;
elsif PSTATE.EL == EL2 then
    R[t] = ID_MMFR4;
elsif PSTATE.EL == EL3 then
    R[t] = ID_MMFR4;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR5, Memory Model Feature Register 5

The ID_MMFR5 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

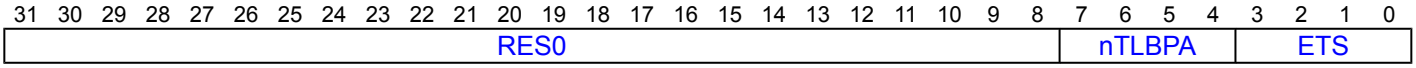
AArch32 System register ID_MMFR5 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR5_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_MMFR5 are UNDEFINED.

Attributes

ID_MMFR5 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

nTLBPA, bits [7:4]

Indicates support for intermediate caching of translation table walks.

The value of this field is an IMPLEMENTATION DEFINED choice of:

nTLBPA	Meaning
0b0000	The intermediate caching of translation table walks might include non-coherent physical translation caches.
0b0001	The intermediate caching of translation table walks does not include non-coherent physical translation caches.

Non-coherent physical translation caches are non-coherent caches of previous valid translation table entries since the last completed relevant TLBI applicable to the PE, where either:

- The caching is indexed by the physical address of the location holding the translation table entry.
- The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry.

All other values are reserved.

FEAT_nTLBPA implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

ETS, bits [3:0]

Indicates support for Enhanced Translation Synchronization.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ETS	Meaning
0b0000	Enhanced Translation Synchronization is not supported.
0b0001	Enhanced Translation Synchronization is not supported.
0b0010	FEAT_ETS2 is implemented.
0b0011	FEAT_ETS3 is implemented.

All other values are reserved.

FEAT_ETS2 implements the functionality identified by the value 0b0010.

FEAT_ETS3 implements the functionality identified by the value 0b0011.

From Armv8.8, the values 0b0000 and 0b0001 are not permitted.

From Armv9.5, the value 0b0010 is not permitted.

Access to this field is **RO**.

Accessing ID_MMFR5

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0011	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
(IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_MMFR5) || boolean IMPLEMENTATION_DEFINED "ID_MMFR5
trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
(IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_MMFR5) || boolean IMPLEMENTATION_DEFINED "ID_MMFR5
trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_MMFR5;
elsif PSTATE.EL == EL2 then
    R[t] = ID_MMFR5;
elsif PSTATE.EL == EL3 then
    R[t] = ID_MMFR5;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_PFR0, Processor Feature Register 0

The ID_PFR0 characteristics are:

Purpose

Gives top-level information about the instruction sets and other features supported by the PE in AArch32 state.

Must be interpreted with ID_PFR1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

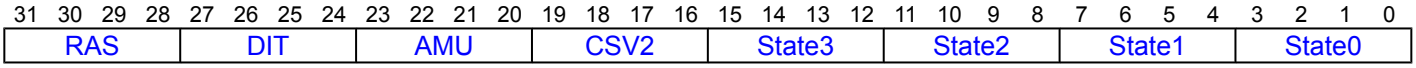
AArch32 System register ID_PFR0 bits [31:0] are architecturally mapped to AArch64 System register ID_PFR0_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_PFR0 are UNDEFINED.

Attributes

ID_PFR0 is a 32-bit register.

Field descriptions



RAS, bits [31:28]

RAS Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RAS	Meaning
0b0000	The RAS Extension is not implemented.
0b0001	The RAS Extension is implemented, FEAT_RAS provides the ESB instruction and the Error synchronization event.
0b0010	FEAT_RASv1p1 implemented. As 0b0001, and adds support for additional ERXMISC<m> System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp Extension.
0b0011	FEAT_RASv2 implemented. As 0b0010, and requires that error records accessed through System registers conform to RAS System Architecture v2.

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0001.

FEAT_RASv1p1 implements the functionality identified by the value 0b0010.

FEAT_RASv2 implements the functionality identified by the value 0b0011.

In Armv8.0 and Armv8.1, the permitted values are 0b0000 and 0b0001.

From Armv8.2, the value 0b0000 is not permitted.

From Armv8.4, if FEAT_DoubleFault is implemented or ERRIDR.NUM is nonzero, the value 0b0001 is not permitted.

Note

When the value of this field is 0b0001, ID_PFR2.RAS_frac indicates whether FEAT_RASv1p1 is implemented.

From Armv8.9, if FEAT_DoubleFault is implemented or ERRIDR_EL1.NUM is nonzero, the value 0b0010 is not permitted.

Access to this field is **RO**.

DIT, bits [27:24]

Data Independent Timing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DIT	Meaning
0b0000	AArch32 does not guarantee constant execution time of any instructions.
0b0001	AArch32 provides the PSTATE.DIT mechanism to guarantee constant execution time of certain instructions.

All other values are reserved.

FEAT_DIT implements the functionality identified by the value 0b0001.

From Armv8.4, the value 0b0000 is not permitted.

Access to this field is **RO**.

AMU, bits [23:20]

Indicates support for Activity Monitors Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AMU	Meaning
0b0000	Activity Monitors Extension is not implemented.
0b0001	FEAT_AMUv1 is implemented.
0b0010	FEAT_AMUv1p1 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters.

All other values are reserved.

FEAT_AMUv1 implements the functionality identified by the value 0b0001.

FEAT_AMUv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0000.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.6, the permitted values are 0b0000, 0b0001, and 0b0010.

Access to this field is **RO**.

CSV2, bits [19:16]

Speculative use of out of context branch targets.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV2	Meaning
0b0000	The implementation does not disclose whether FEAT_CSV2 is implemented.
0b0001	FEAT_CSV2 is implemented, but FEAT_CSV2_1p1 is not implemented.
0b0010	FEAT_CSV2_1p1 is implemented.

All other values are reserved.

FEAT_CSV2 implements the functionality identified by the value 0b0001.

FEAT_CSV2_1p1 implements the functionality identified by the value 0b0010.

From Armv8.5, the permitted values are 0b0001 and 0b0010.

Access to this field is **RO**.

State3, bits [15:12]

T32EE instruction set support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State3	Meaning
0b0000	Not implemented.
0b0001	T32EE instruction set implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

State2, bits [11:8]

Jazelle extension support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State2	Meaning
0b0000	Not implemented.
0b0001	Jazelle extension implemented, without clearing of JOSCR.CV on exception entry.
0b0010	Jazelle extension implemented, with clearing of JOSCR.CV on exception entry.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

State1, bits [7:4]

T32 instruction set support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State1	Meaning
0b0000	T32 instruction set not implemented.
0b0001	T32 encodings before the introduction of Thumb-2 technology implemented: • All instructions are 16-bit. • A BL or BLX is a pair of 16-bit instructions. • 32-bit instructions other than BL and BLX cannot be encoded.
0b0011	T32 encodings after the introduction of Thumb-2 technology implemented, for all 16-bit and 32-bit T32 basic instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Access to this field is **RO**.

State0, bits [3:0]

A32 instruction set support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State0	Meaning
0b0000	A32 instruction set not implemented.
0b0001	A32 instruction set implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Accessing ID_PFR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_PFR0;
elsif PSTATE.EL == EL2 then
    R[t] = ID_PFR0;
elsif PSTATE.EL == EL3 then
    R[t] = ID_PFR0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_PFR1, Processor Feature Register 1

The ID_PFR1 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with ID_PFR0.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

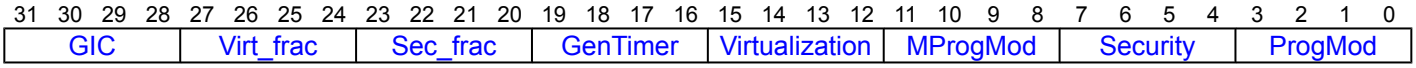
AArch32 System register ID_PFR1 bits [31:0] are architecturally mapped to AArch64 System register ID_PFR1_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_PFR1 are UNDEFINED.

Attributes

ID_PFR1 is a 32-bit register.

Field descriptions



GIC, bits [31:28]

System register GIC CPU interface.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GIC	Meaning
0b0000	GIC CPU interface system registers not implemented.
0b0001	System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.
0b0011	System register interface to version 4.1 of the GIC CPU interface is supported.

All other values are reserved.

Access to this field is **RO**.

Virt_frac, bits [27:24]

Virtualization fractional field. When the Virtualization field is 0b0000, determines the support for Virtualization Extensions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Virt_frac	Meaning
0b0000	No Virtualization Extensions are implemented.
0b0001	The following Virtualization Extensions are implemented: • The SCR.SIF bit, if EL3 is implemented. • The modifications to the SCR.AW and SCR.FW bits described in the Virtualization Extensions, if EL3 is implemented. • The MSR (banked register) and MRS (banked register) instructions. • The ERET instruction.

All other values are reserved.

In Armv8-A, the permitted values are:

- `0b0000` when EL2 is implemented.
- `0b0001` when EL2 is not implemented.

This field is valid only when the value of ID_PFR1.Virtualization is 0, otherwise it holds the value `0b0000`.

Note

The ID_ISAR registers do not identify whether the instructions added by the Virtualization Extensions are implemented.

Access to this field is **RO**.

Sec_frac, bits [23:20]

Security fractional field. When the Security field is `0b0000`, determines the support for Security Extensions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Sec_frac	Meaning
`0b0000`	No Security Extensions are implemented.
`0b0001`	The following Security Extensions are implemented: • The VBAR register. • The TTBCR.PD0 and TTBCR.PD1 bits.
`0b0010`	As for `0b0001`, and the ability to access Secure or Non-secure physical memory is supported.

All other values are reserved.

In Armv8-A, the permitted values are:

- `0b0000` when EL3 is implemented.
- `0b0001` or `0b0010` when EL3 is not implemented.

This field is valid only when the value of ID_PFR1.Security is 0, otherwise it holds the value `0b0000`.

Access to this field is **RO**.

GenTimer, bits [19:16]

Generic Timer support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GenTimer	Meaning
`0b0000`	Generic Timer is not implemented.
`0b0001`	Generic Timer is implemented.
`0b0010`	Generic Timer is implemented, and also includes support for CNTHCTL.EVNTIS and CNTKCTL.EVNTIS fields, and CNTPCTSS and CNTVCTSS counter views.

All other values are reserved.

FEAT_ECV implements the functionality identified by the value `0b0010`.

In Armv8.0, the only permitted value is `0b0001`.

From Armv8.6, the only permitted value is `0b0010`.

Access to this field is **RO**.

Virtualization, bits [15:12]

Virtualization support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Virtualization	Meaning
0b0000	EL2, Hyp mode, and the HVC instruction not implemented.
0b0001	EL2, Hyp mode, the HVC instruction, and all the features described by Virt_frac == 0b0001 implemented.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0001 when EL2 is implemented.

In an implementation that includes EL2, if EL2 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

Note

The ID_ISARs do not identify whether the HVC instruction is implemented.

Access to this field is **RO**.

MProgMod, bits [11:8]

M-profile programmers' model support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MProgMod	Meaning
0b0000	Not supported.
0b0010	Support for two-stack programmers' model.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

Security, bits [7:4]

Security support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Security	Meaning
0b0000	EL3, Monitor mode, and the SMC instruction not implemented.
0b0001	EL3, Monitor mode, the SMC instruction, and all the features described by Sec_frac == 0b0001 implemented.
0b0010	As for 0b0001, and adds the ability to set the NSACR.RFR bit. Not permitted in Armv8 as the NSACR.RFR bit is RES0.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL3 is not implemented.
- 0b0001 when EL3 is implemented.

In an implementation that includes EL3, if EL3 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

Access to this field is **RO**.

ProgMod, bits [3:0]

Support for the standard programmers' model for ARMv4 and later. Model must support User, FIQ, IRQ, Supervisor, Abort, Undefined, and System modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ProgMod	Meaning
0b0000	Not supported.
0b0001	Supported.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is **RO**.

Accessing ID_PFR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_PFR1;
elsif PSTATE.EL == EL2 then
    R[t] = ID_PFR1;
elsif PSTATE.EL == EL3 then
    R[t] = ID_PFR1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_PFR2, Processor Feature Register 2

The ID_PFR2 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with ID_PFR0 and ID_PFR1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

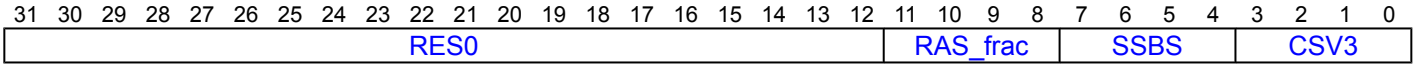
AArch32 System register ID_PFR2 bits [31:0] are architecturally mapped to AArch64 System register ID_PFR2_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ID_PFR2 are UNDEFINED.

Attributes

ID_PFR2 is a 32-bit register.

Field descriptions



Bits [31:12]

Reserved, RES0.

RAS_frac, bits [11:8]

RAS Extension fractional field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RAS_frac	Meaning
0b0000	If ID_PFR0.RAS == 0b0001, support for the Reliability, Availability, and Serviceability Extension is implemented.
0b0001	If ID_PFR0.RAS == 0b0001, as 0b0000 and adds support for additional ERXMISC<m> System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp Extension.

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0000.

FEAT_RASv1p1 implements the functionality identified by the value 0b0001.

This field is valid only if ID_PFR0.RAS == 0b0001.

Access to this field is **RO**.

SSBS, bits [7:4]

Speculative Store Bypassing controls in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SSBS	Meaning
0b0000	AArch32 provides no mechanism to control the use of Speculative Store Bypassing.
0b0001	AArch32 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

All other values are reserved.

Access to this field is **RO**.

CSV3, bits [3:0]

Speculative use of faulting data.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV3	Meaning
0b0000	This PE does not disclose whether data loaded or read from a register under speculation where the data load or register read would not be permitted architecturally, can be used by instructions newer than the load or register read in a manner that allows the value of the inaccessible data to be recovered by code architecturally executed.
0b0001	Data loaded or read from a register under speculation where the data load or register read would not be permitted architecturally, cannot be used by instructions newer than the load or register read in a manner that allows the value of the inaccessible data to be recovered by code architecturally executed.

All other values are reserved.

FEAT_CSV3 implements the functionality identified by the value 0b0001.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

If FEAT_E0PD is implemented, FEAT_CSV3 must be implemented.

Access to this field is **RO**.

Accessing ID_PFR2

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ID_PFR2;
elsif PSTATE.EL == EL2 then
    R[t] = ID_PFR2;
elsif PSTATE.EL == EL3 then
    R[t] = ID_PFR2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

IFAR, Instruction Fault Address Register

The IFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Prefetch Abort exception.

Configuration

This register is banked between IFAR and IFAR_S and IFAR_NS.

AArch32 System register IFAR bits [31:0] are architecturally mapped to AArch64 System register FAR_EL1[63:32].

AArch32 System register IFAR bits [31:0] (IFAR_S) are architecturally mapped to AArch32 System register HIFAR[31:0] when EL2 is implemented and EL3 is implemented.

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to IFAR are UNDEFINED.

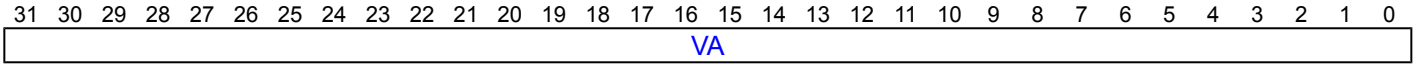
Attributes

IFAR is a 32-bit register.

This register has the following instances:

- IFAR, when EL3 is not implemented or FEAT_AA64 is implemented.
- IFAR_S, when FEAT_AA32EL3 is implemented.
- IFAR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



VA, bits [31:0]

VA of faulting address of synchronous Prefetch Abort exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing IFAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0110	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T6
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T6 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = IFAR_NS;
    else
        R[t] = IFAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = IFAR_NS;
    else
        R[t] = IFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = IFAR_S;
    else
        R[t] = IFAR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0110	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T6
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T6 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        IFAR_NS = R[t];
    else
        IFAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        IFAR_NS = R[t];
    else
        IFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        IFAR_S = R[t];
    else
        IFAR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

IFSR, Instruction Fault Status Register

The IFSR characteristics are:

Purpose

Holds status information about the last instruction fault.

Configuration

This register is banked between IFSR and IFSR_S and IFSR_NS.

AArch32 System register IFSR bits [31:0] are architecturally mapped to AArch64 System register IFSR32_EL2[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to IFSR are UNDEFINED.

The current translation table format determines which format of the register is used.

Attributes

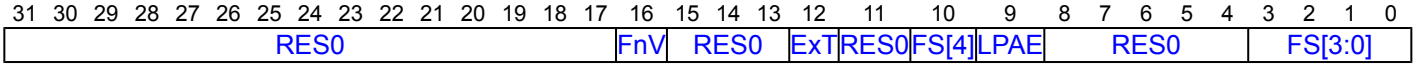
IFSR is a 32-bit register.

This register has the following instances:

- IFSR, when EL3 is not implemented or FEAT_AA64 is implemented.
- IFSR_S, when FEAT_AA32EL3 is implemented.
- IFSR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When TTBCR.EAE == 0:



Bits [31:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	IFAR is valid.
0b1	IFAR is not valid, and holds an UNKNOWN value.

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault Status bits. Bits [10] and [3:0] are interpreted together.

FS	Meaning	Applies when
0b00001	PC alignment fault.	
0b00010	Debug exception.	
0b00011	Access flag fault, level 1.	
0b00101	Translation fault, level 1.	
0b00110	Access flag fault, level 2.	
0b00111	Translation fault, level 2.	
0b01000	Synchronous External abort, not on translation table walk.	
0b01001	Domain fault, level 1.	
0b01011	Domain fault, level 2.	
0b01100	Synchronous External abort, on translation table walk, level 1.	
0b01101	Permission fault, level 1.	
0b01110	Synchronous External abort, on translation table walk, level 2.	
0b01111	Permission fault, level 2.	
0b10000	TLB conflict abort.	
0b10100	IMPLEMENTATION DEFINED fault (Lockdown fault).	
0b11001	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b11100	Synchronous parity or ECC error on translation table walk, level 1.	When FEAT_RAS is not implemented
0b11110	Synchronous parity or ECC error on translation table walk, level 2.	When FEAT_RAS is not implemented

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Short-descriptor translation table lookup'.

The FS field is split as follows:

- FS[4] is IFSR[10].
- FS[3:0] is IFSR[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

LPAE	Meaning
0b0	Using the Short-descriptor translation table formats.
0b1	Using the Long-descriptor translation table formats.

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

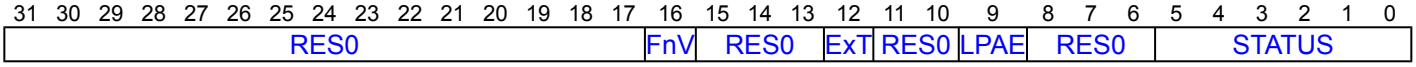
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When TTBCR.EAE == 1:



Bits [31:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	IFAR is valid.
0b1	IFAR is not valid, and holds an UNKNOWN value.

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

LPAE	Meaning
0b0	Using the Short-descriptor translation table formats.
0b1	Using the Long-descriptor translation table formats.

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status bits. Possible values of this field are:

STATUS	Meaning	Applies when
0b000000	Address size fault in translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk.	
0b010101	Synchronous External abort on translation table walk, level 1.	
0b010110	Synchronous External abort on translation table walk, level 2.	
0b010111	Synchronous External abort on translation table walk, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk, level 3.	When FEAT_RAS is not implemented
0b100001	PC alignment fault.	
0b100010	Debug exception.	
0b110000	TLB conflict abort.	

All other values are reserved.

When FEAT_RAS is implemented, 0b011000, 0b011101, 0b011110, and 0b011111 are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing IFSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = IFSR_NS;
    else
        R[t] = IFSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = IFSR_NS;
    else
        R[t] = IFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = IFSR_S;
    else
        R[t] = IFSR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        IFSR_NS = R[t];
    else
        IFSR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        IFSR_NS = R[t];
    else
        IFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        IFSR_S = R[t];
    else
        IFSR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ISR, Interrupt Status Register

The ISR characteristics are:

Purpose

Shows the pending status of the IRQ and FIQ interrupts and the SError exceptions.

Configuration

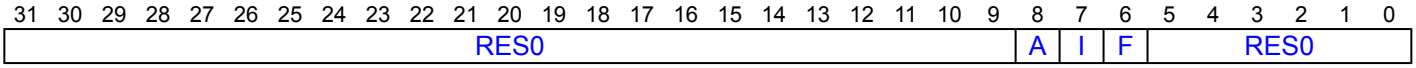
AArch32 System register ISR bits [31:0] are architecturally mapped to AArch64 System register ISR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ISR are UNDEFINED.

Attributes

ISR is a 32-bit register.

Field descriptions



Bits [31:9]

Reserved, RES0.

A, bit [8]

SError exception pending bit:

A	Meaning
0b0	No pending SError exception.
0b1	An SError exception is pending.

If all of the following apply then this field shows the pending status of virtual SError exceptions:

- EL2 is implemented and enabled in the current Security state.
- Any of the following apply:
  - EL2 is using AArch64 and HCR_EL2.AMO is 1.
  - EL2 is using AArch64, FEAT_DoubleFault2 is implemented, and the Effective value of HCRX_EL2.TMEA is 1.
  - EL2 is using AArch32 and HCR.AMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical SError exceptions.

If the SError exception is edge-triggered, this field is cleared to zero when the physical SError exception is taken.

I, bit [7]

IRQ pending bit. Indicates whether an IRQ interrupt is pending:

I	Meaning
0b0	No pending IRQ.
0b1	An IRQ interrupt is pending.

If all of the following apply then this field shows the pending status of virtual IRQ interrupts:

- EL2 is implemented and enabled in the current Security state.
- Any of the following apply:
    - EL2 is using AArch64 and HCR_EL2.IMO is 1.
    - EL2 is using AArch32 and HCR.IMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical IRQ interrupts.

F, bit [6]

FIQ pending bit. Indicates whether an FIQ interrupt is pending.

F	Meaning
0b0	No pending FIQ.
0b1	An FIQ interrupt is pending.

If all of the following apply then this field shows the pending status of virtual FIQ interrupts:

- EL2 is implemented and enabled in the current Security state.
- Any of the following apply:
    - EL2 is using AArch64 and HCR_EL2.FMO is 1.
    - EL2 is using AArch32 and HCR.FMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical FIQ interrupts.

Bits [5:0]

Reserved, RES0.

Accessing ISR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = ISR;
elsif PSTATE.EL == EL2 then
    R[t] = ISR;
elsif PSTATE.EL == EL3 then
    R[t] = ISR;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ITLBIALL, Instruction TLB Invalidate All

The ITLBIALL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from instruction TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
  - Would be required for the EL1&0 translation regime.
  - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2, and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the Non-secure PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ITLBIALL are UNDEFINED.

Attributes

ITLBIALL is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing ITLBIALL

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.ITLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_ExcludeXS);
        else
            AArch32.ITLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_AllAttr);
elsif PSTATE.EL == EL2 then
    AArch32.ITLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    AArch32.ITLBI_ALL(SecurityStateAtEL(EL3), Regime_EL30, Broadcast_NSH, TLBI_AllAttr);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ITLBIASID, Instruction TLB Invalidate by ASID match

The ITLBIASID characteristics are:

Purpose

Invalidate all cached copies of translation table entries from instruction TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

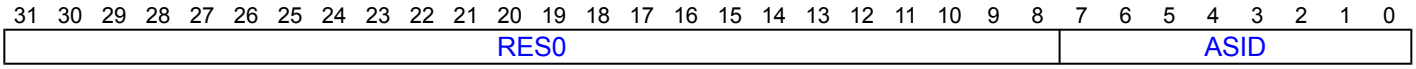
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ITLBIASID are UNDEFINED.

Attributes

ITLBIASID is a 32-bit System instruction.

Field descriptions



Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

Executing ITLBIASID

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0101	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.ITLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, R[t]);
        else
            AArch32.ITLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.ITLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBI_AllAttr,
R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.ITLBI_ASID(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_NSH,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ITLBIMVA, Instruction TLB Invalidate by VA

The ITLBIMVA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from instruction TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

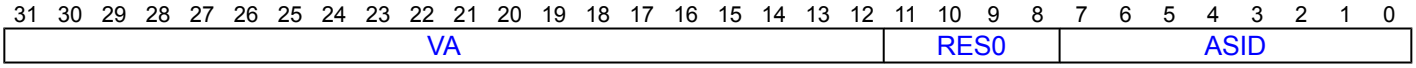
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to ITLBIMVA are UNDEFINED.

Attributes

ITLBIMVA is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this operation, regardless of the value of the ASID field.

Executing ITLBIMVA

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.ITLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.ITLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.ITLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.ITLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

JIDR, Jazelle ID Register

The JIDR characteristics are:

Purpose

A Jazelle register, which identified the Jazelle architecture version.

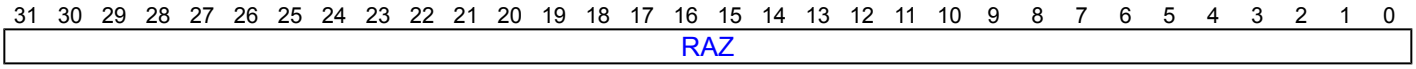
Configuration

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to JIDR are UNDEFINED.

Attributes

JIDR is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RAZ.

Accessing JIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b111	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JIDR UNDEFINED at EL0" then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HCR_EL2.TID0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HCR.TID0
== '1' then
        AArch32.TakeHypTrapException(0x05);
    else
        R[t] = JIDR;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HCR_EL2.TID0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID0 ==
'1' then
        AArch32.TakeHypTrapException(0x05);
    else
        R[t] = JIDR;
elsif PSTATE.EL == EL2 then
    R[t] = JIDR;
elsif PSTATE.EL == EL3 then
    R[t] = JIDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

JMCR, Jazelle Main Configuration Register

The JMCR characteristics are:

Purpose

A Jazelle register, which provides control of the Jazelle extension.

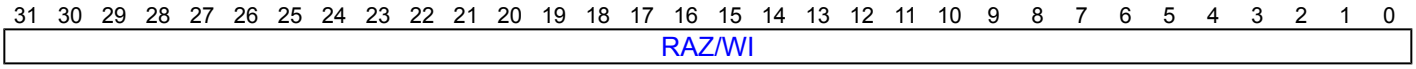
Configuration

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to JMCR are UNDEFINED.

Attributes

JMCR is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RAZ/WI.

Accessing JMCR

For accesses from EL0 it is IMPLEMENTATION DEFINED whether the register is RW or UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b111	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JMCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        R[t] = JMCR;
elsif PSTATE.EL == EL1 then
    R[t] = JMCR;
elsif PSTATE.EL == EL2 then
    R[t] = JMCR;
elsif PSTATE.EL == EL3 then
    R[t] = JMCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1110	0b111	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JMCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        return;
elsif PSTATE.EL == EL1 then
    return;
elsif PSTATE.EL == EL2 then
    return;
elsif PSTATE.EL == EL3 then
    return;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

JOSCR, Jazelle OS Control Register

The JOSCR characteristics are:

Purpose

A Jazelle register, which provides operating system control of the Jazelle Extension.

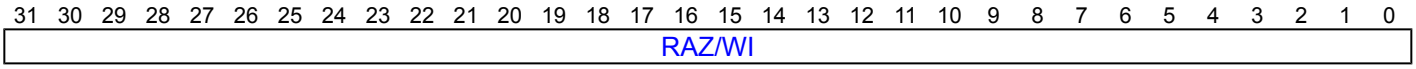
Configuration

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to JOSCR are UNDEFINED.

Attributes

JOSCR is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RAZ/WI.

Accessing JOSCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b111	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JOSCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        R[t] = JOSCR;
elsif PSTATE.EL == EL1 then
    R[t] = JOSCR;
elsif PSTATE.EL == EL2 then
    R[t] = JOSCR;
elsif PSTATE.EL == EL3 then
    R[t] = JOSCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1110	0b111	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JOSCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        return;
elsif PSTATE.EL == EL1 then
    return;
elsif PSTATE.EL == EL2 then
    return;
elsif PSTATE.EL == EL3 then
    return;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR0, Memory Attribute Indirection Register 0

The MAIR0 characteristics are:

Purpose

Along with MAIR1, provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations.

AttrIndx[2] indicates the MAIR register to be used:

- When AttrIndx[2] is 0, MAIR0 is used.
- When AttrIndx[2] is 1, MAIR1 is used.

Configuration

This register is banked between MAIR0 and MAIR0_S and MAIR0_NS.

AArch32 System register MAIR0 bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL1[31:0] when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register MAIR0 bits [31:0] are architecturally mapped to AArch32 System register PRRR[31:0] when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register MAIR0 bits [31:0] (MAIR0_NS) are architecturally mapped to AArch32 System register PRRR[31:0] (PRRR_NS) when EL3 is using AArch32.

AArch32 System register MAIR0 bits [31:0] (MAIR0_S) are architecturally mapped to AArch32 System register PRRR[31:0] (PRRR_S) when EL3 is using AArch32.

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to MAIR0 are UNDEFINED.

MAIR0 and PRRR are the same register, with a different view depending on the value of TTBCR.EAE:

- When it is set to 0, the register is as described in PRRR.
- When it is set to 1, the register is as described in MAIR0.

When EL3 is using AArch32, write access to MAIR0(S) is disabled when the **CP15SDISABLE** signal is asserted HIGH.

Attributes

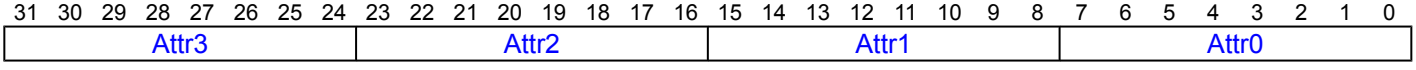
MAIR0 is a 32-bit register.

This register has the following instances:

- MAIR0, when EL3 is not implemented or FEAT_AA64 is implemented.
- MAIR0_S, when FEAT_AA32EL3 is implemented.
- MAIR0_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When TTBCR.EAE == 1:



Attr<n>, bits [8n+7:8n], for n = 3 to 0

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIndx[2:0] gives the value of <n> in Attr<n>.
- AttrIndx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

Attr<n>[7:4]	Meaning
0b0000	Device memory. See encoding of Attr<n>[3:0] for the type of Device memory.
0b00RW, RW not 0b00	Normal memory, Outer Write-Through Transient.
0b0100	Normal memory, Outer Non-cacheable.
0b01RW, RW not 0b00	Normal memory, Outer Write-Back Transient.
0b10RW	Normal memory, Outer Write-Through Non-transient.
0b11RW	Normal memory, Outer Write-Back Non-transient.

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

Attr<n>[3:0]	Meaning when Attr<n>[7:4] is 0b0000	Meaning when Attr<n>[7:4] is not 0b0000
0b0000	Device-nGnRnE memory	UNPREDICTABLE
0b00RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Through Transient
0b0100	Device-nGnRE memory	Normal memory, Inner Non-cacheable
0b01RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Back Transient
0b1000	Device-nGRE memory	Normal memory, Inner Write-Through Non-transient (RW=0b00)
0b10RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Through Non-transient
0b1100	Device-GRE memory	Normal memory, Inner Write-Back Non-transient (RW=0b00)
0b11RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Back Non-transient

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<n> fields have the following meanings:

R or W	Meaning
0b0	No Allocate
0b1	Allocate

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            R[t] = MAIR0_NS;
        else
            R[t] = PRRR_NS;
    else
        if TTBCR.EAE == '1' then
            R[t] = MAIR0;
        else
            R[t] = PRRR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            R[t] = MAIR0_NS;
        else
            R[t] = PRRR_NS;
    else
        if TTBCR.EAE == '1' then
            R[t] = MAIR0;
        else
            R[t] = PRRR;
elsif PSTATE.EL == EL3 then
    if TTBCR.EAE == '1' then
        if SCR.NS == '0' then
            R[t] = MAIR0_S;
        else
            R[t] = MAIR0_NS;
    else
        if SCR.NS == '0' then
            R[t] = PRRR_S;
        else
            R[t] = PRRR_NS;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR0_NS = R[t];
        else
            PRRR_NS = R[t];
    else
        if TTBCR.EAE == '1' then
            MAIR0 = R[t];
        else
            PRRR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR0_NS = R[t];
        else
            PRRR_NS = R[t];
    else
        if TTBCR.EAE == '1' then
            MAIR0 = R[t];
        else
            PRRR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if TTBCR.EAE == '1' then
            if SCR.NS == '0' then
                MAIR0_S = R[t];
            else
                MAIR0_NS = R[t];
        else
            if SCR.NS == '0' then
                PRRR_S = R[t];
            else
                PRRR_NS = R[t];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR1, Memory Attribute Indirection Register 1

The MAIR1 characteristics are:

Purpose

Along with MAIR0, provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations.

AttrIndx[2] indicates the MAIR register to be used:

- When AttrIndx[2] is 0, MAIR0 is used.
- When AttrIndx[2] is 1, MAIR1 is used.

Configuration

This register is banked between MAIR1 and MAIR1_S and MAIR1_NS.

AArch32 System register MAIR1 bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL1[63:32] when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register MAIR1 bits [31:0] are architecturally mapped to AArch32 System register NMRR[31:0] when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register MAIR1 bits [31:0] (MAIR1_NS) are architecturally mapped to AArch32 System register NMRR[31:0] (NMRR_NS) when EL3 is using AArch32.

AArch32 System register MAIR1 bits [31:0] (MAIR1_S) are architecturally mapped to AArch32 System register NMRR[31:0] (NMRR_S) when EL3 is using AArch32.

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to MAIR1 are UNDEFINED.

MAIR1 and NMRR are the same register, with a different view depending on the value of TTBCR.EAE:

- When it is set to 0, the register is as described in NMRR.
- When it is set to 1, the register is as described in MAIR1.

When EL3 is using AArch32, write access to MAIR1(S) is disabled when the **CP15SDISABLE** signal is asserted HIGH.

Attributes

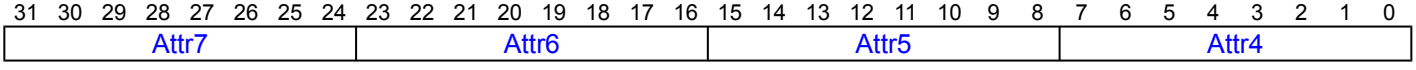
MAIR1 is a 32-bit register.

This register has the following instances:

- MAIR1, when EL3 is not implemented or FEAT_AA64 is implemented.
- MAIR1_S, when FEAT_AA32EL3 is implemented.
- MAIR1_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When TTBCR.EAE == 1:



Attr<n>, bits [8(n-4)+7:8(n-4)], for n = 7 to 4

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIndx[2:0] gives the value of <n> in Attr<n>.
- AttrIndx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

Attr<n>[7:4]	Meaning
0b0000	Device memory. See encoding of Attr<n>[3:0] for the type of Device memory.
0b00RW, RW not 0b00	Normal memory, Outer Write-Through Transient.
0b0100	Normal memory, Outer Non-cacheable.
0b01RW, RW not 0b00	Normal memory, Outer Write-Back Transient.
0b10RW	Normal memory, Outer Write-Through Non-transient.
0b11RW	Normal memory, Outer Write-Back Non-transient.

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

Attr<n>[3:0]	Meaning when Attr<n>[7:4] is 0b0000	Meaning when Attr<n>[7:4] is not 0b0000
0b0000	Device-nGnRnE memory	UNPREDICTABLE
0b00RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Through Transient
0b0100	Device-nGnRE memory	Normal memory, Inner Non-cacheable
0b01RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Back Transient
0b1000	Device-nGRE memory	Normal memory, Inner Write-Through Non-transient (RW=0b00)
0b10RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Through Non-transient
0b1100	Device-GRE memory	Normal memory, Inner Write-Back Non-transient (RW=0b00)
0b11RW, RW not 0b00	UNPREDICTABLE	Normal memory, Inner Write-Back Non-transient

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<n> fields have the following meanings:

R or W	Meaning
0b0	No Allocate
0b1	Allocate

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            R[t] = MAIR1_NS;
        else
            R[t] = NMRR_NS;
    else
        if TTBCR.EAE == '1' then
            R[t] = MAIR1;
        else
            R[t] = NMRR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            R[t] = MAIR1_NS;
        else
            R[t] = NMRR_NS;
    else
        if TTBCR.EAE == '1' then
            R[t] = MAIR1;
        else
            R[t] = NMRR;
elsif PSTATE.EL == EL3 then
    if TTBCR.EAE == '1' then
        if SCR.NS == '0' then
            R[t] = MAIR1_S;
        else
            R[t] = MAIR1_NS;
    else
        if SCR.NS == '0' then
            R[t] = NMRR_S;
        else
            R[t] = NMRR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR1_NS = R[t];
        else
            NMRR_NS = R[t];
    else
        if TTBCR.EAE == '1' then
            MAIR1 = R[t];
        else
            NMRR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR1_NS = R[t];
        else
            NMRR_NS = R[t];
    else
        if TTBCR.EAE == '1' then
            MAIR1 = R[t];
        else
            NMRR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if TTBCR.EAE == '1' then
            if SCR.NS == '0' then
                MAIR1_S = R[t];
            else
                MAIR1_NS = R[t];
        else
            if SCR.NS == '0' then
                NMRR_S = R[t];
            else
                NMRR_NS = R[t];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MIDR, Main ID Register

The MIDR characteristics are:

Purpose

Provides identification information for the PE, including an implementer code for the device and a device ID number.

Configuration

AArch32 System register MIDR bits [31:0] are architecturally mapped to AArch64 System register MIDR_EL1[31:0].

AArch32 System register MIDR bits [31:0] are architecturally mapped to External register MIDR_EL1[31:0].

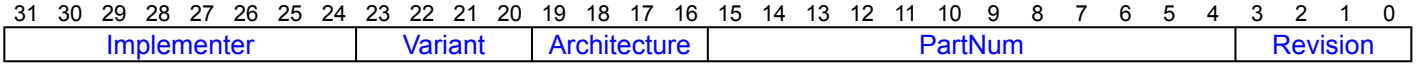
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to MIDR are UNDEFINED.

Some fields of the MIDR are IMPLEMENTATION DEFINED. For more information about the values of these fields for a particular Armv8 implementation, and any implementation-specific significance of these values, see the product documentation.

Attributes

MIDR is a 32-bit register.

Field descriptions



Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Implementer	Meaning
0x00	Reserved for software use.
0x41	Arm Limited.
0x42	Broadcom Corporation.
0x43	Cavium Inc.
0x44	Digital Equipment Corporation.
0x46	Fujitsu Ltd.
0x49	Infineon Technologies AG.
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation.
0x50	Applied Micro Circuits Corporation.
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation.
0xC0	Ampere Computing.

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

Access to this field is **RO**.

Variant, bits [23:20]

Variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Architecture, bits [19:16]

Architecture version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Architecture	Meaning
0b0001	Armv4.
0b0010	Armv4T.
0b0011	Armv5 (obsolete).
0b0100	Armv5T.
0b0101	Armv5TE.
0b0110	Armv5TEJ.
0b0111	Armv6.
0b1111	Architectural features are individually identified in the ID_* registers.

All other values are reserved.

Access to this field is **RO**.

PartNum, bits [15:4]

Primary Part Number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [3:0]

Revision number for the device.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        R[t] = VPIDR_EL2<31:0>;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
        R[t] = VPIDR;
    else
        R[t] = MIDR;
elsif PSTATE.EL == EL2 then
    R[t] = MIDR;
elsif PSTATE.EL == EL3 then
    R[t] = MIDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPIDR, Multiprocessor Affinity Register

The MPIDR characteristics are:

Purpose

In a multiprocessor system, provides an additional PE identification mechanism.

Configuration

AArch32 System register MPIDR bits [31:0] are architecturally mapped to AArch64 System register MPIDR_EL1[31:0].

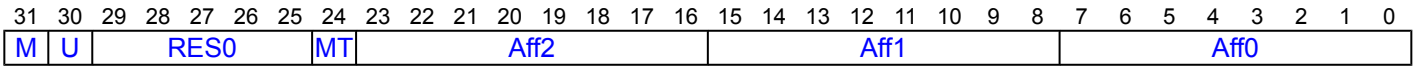
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to MPIDR are UNDEFINED.

In a uniprocessor system, Arm recommends that each Aff<n> field of this register returns a value of 0.

Attributes

MPIDR is a 32-bit register.

Field descriptions



M, bit [31]

Indicates whether this implementation includes the functionality introduced by the Armv7 Multiprocessing Extensions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

M	Meaning
0b0	This implementation does not include the Armv7 Multiprocessing Extensions functionality.
0b1	This implementation includes the Armv7 Multiprocessing Extensions functionality.

Access to this field is **RAO/WI**.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Access to this field is **RO**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is **RO**.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        R[t] = VMPIDR_EL2<31:0>;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
        R[t] = VMPIDR;
    else
        R[t] = MPIDR;
elsif PSTATE.EL == EL2 then
    R[t] = MPIDR;
elsif PSTATE.EL == EL3 then
    R[t] = MPIDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVBAR, Monitor Vector Base Address Register

The MVBAR characteristics are:

Purpose

When EL3 is implemented and can use AArch32, holds the vector base address for any exception that is taken to Monitor mode.

Secure software must program the MVBAR with the required initial value as part of the PE boot sequence.

Configuration

This register is present only when FEAT_AA32EL3 is implemented. Otherwise, direct accesses to MVBAR are UNDEFINED.

It is IMPLEMENTATION DEFINED whether MVBAR[0] has a fixed value and ignored writes, or takes the last value written to it.

On a Warm reset into EL3 using AArch32, the reset value of MVBAR is an IMPLEMENTATION DEFINED choice between the following:

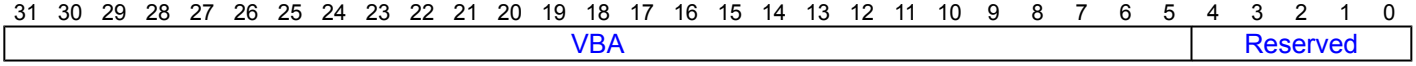
- MVBAR[31:5] = an IMPLEMENTATION DEFINED value, which might be UNKNOWN, MVBAR[4:1] = RES0, and MVBAR[0] = 0.
- MVBAR[31:1] = an IMPLEMENTATION DEFINED value that is bits[31:1] of the AArch32 reset address, and MVBAR[0] = 1.

Attributes

MVBAR is a 32-bit register.

Field descriptions

When programmed with a vector base address:



VBA, bits [31:5]

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

Reserved, bits [4:0]

Reserved, see Configurations.

Accessing MVBAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL3) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsHighestEL(EL1) then
        R[t] = RVBAR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsHighestEL(EL2) then
        R[t] = RVBAR;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    R[t] = MVBAR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL3) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        MVBAR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR0, Media and VFP Feature Register 0

The MVFR0 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Must be interpreted with MVFR1 and MVFR2.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register MVFR0 bits [31:0] are architecturally mapped to AArch64 System register MVFR0_EL1[31:0].

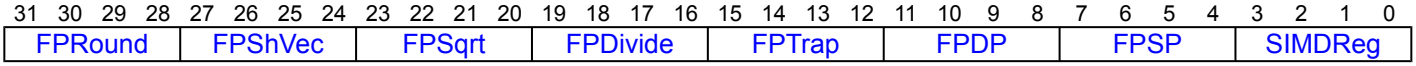
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to MVFR0 are UNDEFINED.

Implemented only if the implementation includes Advanced SIMD and floating-point instructions.

Attributes

MVFR0 is a 32-bit register.

Field descriptions



FPRound, bits [31:28]

Floating-Point Rounding modes. Indicates whether the floating-point implementation provides support for rounding modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPRound	Meaning
0b0000	Not implemented, or only Round to Nearest mode supported, except that Round towards Zero mode is supported for VCVT instructions that always use that rounding mode regardless of the FPSCR setting.
0b0001	All rounding modes supported.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is **RO**.

FPShVec, bits [27:24]

Short Vectors. Indicates whether the floating-point implementation provides support for the use of short vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPShVec	Meaning
0b0000	Short vectors not supported.
0b0001	Short vector operation supported.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0000`.

Access to this field is **RO**.

FPSqrt, bits [23:20]

Square Root. Indicates whether the floating-point implementation provides support for the ARMv6 VFP square root operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPSqrt	Meaning
`0b0000`	Not supported in hardware.
`0b0001`	Supported.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

The VSQRT.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VSQRT.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

Access to this field is **RO**.

FPDivide, bits [19:16]

Indicates whether the floating-point implementation provides support for VFP divide operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPDivide	Meaning
`0b0000`	Not supported in hardware.
`0b0001`	Supported.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

The VDIV.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VDIV.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

Access to this field is **RO**.

FPTrap, bits [15:12]

Floating-Point Exception Trapping. Indicates whether the floating-point implementation provides support for exception trapping.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPTrap	Meaning
`0b0000`	Not supported.
`0b0001`	Supported.

All other values are reserved.

A value of `0b0001` indicates that, when the corresponding trap is enabled, a floating-point exception generates an exception.

Access to this field is **RO**.

FPDP, bits [11:8]

Floating-Point Double-Precision. Indicates whether the floating-point implementation provides support for double-precision operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPDP	Meaning
0b0000	Not supported in hardware.
0b0001	Supported, VFPv2.
0b0010	Supported, VFPv3, VFPv4, or Armv8. VFPv3 and Armv8 add an instruction to load a double-precision floating-point constant, and conversions between double-precision and fixed-point values.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP double-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F64 is only available if the Square root field is 0b0001.
- VDIV.F64 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the single-precision field is nonzero.

Access to this field is **RO**.

FPSP, bits [7:4]

Floating-Point Single-Precision. Indicates whether the floating-point implementation provides support for single-precision operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPSP	Meaning
0b0000	Not supported in hardware.
0b0001	Supported, VFPv2.
0b0010	Supported, VFPv3 or VFPv4. VFPv3 adds an instruction to load a single-precision floating-point constant, and conversions between single-precision and fixed-point values.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP single-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F32 is only available if the Square root field is 0b0001.
- VDIV.F32 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the double-precision field is nonzero.

Access to this field is **RO**.

SIMDReg, bits [3:0]

Advanced SIMD registers. Indicates whether the Advanced SIMD and floating-point implementation provides support for the Advanced SIMD and floating-point register bank.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDReg	Meaning
0b0000	The implementation has no Advanced SIMD and floating-point support.
0b0001	The implementation includes floating-point support with 16 x 64-bit registers.
0b0010	The implementation includes Advanced SIMD and floating-point support with 32 x 64-bit registers.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

Access to this field is **RO**.

Accessing MVFR0

Accesses to this register use the following encodings in the System register encoding space:

```
VMRS{<c>}{<q>} <Rt>, <spec_reg>
```

reg
0b0111

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = MVFR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS ==
'1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = MVFR0;
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        R[t] = MVFR0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR1, Media and VFP Feature Register 1

The MVFR1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Must be interpreted with MVFR0 and MVFR2.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register MVFR1 bits [31:0] are architecturally mapped to AArch64 System register MVFR1_EL1[31:0].

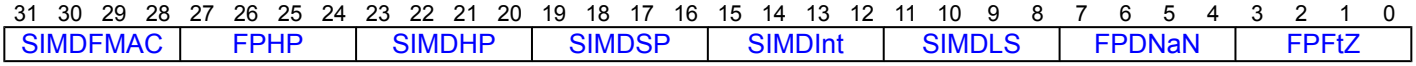
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to MVFR1 are UNDEFINED.

Implemented only if the implementation includes Advanced SIMD and floating-point instructions.

Attributes

MVFR1 is a 32-bit register.

Field descriptions



SIMDFMAC, bits [31:28]

Advanced SIMD Fused Multiply-Accumulate. Indicates whether the Advanced SIMD implementation provides fused multiply accumulate instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDFMAC	Meaning
0b0000	Not implemented.
0b0001	Implemented.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

The Advanced SIMD and floating-point implementations must provide the same level of support for these instructions.

Access to this field is **RO**.

FPHP, bits [27:24]

Floating-Point Half-Precision. Indicates the level of half-precision floating-point support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPHP	Meaning
0b0000	Not supported.
0b0001	Floating-point half-precision conversion instructions are supported for conversion between single-precision and half-precision.
0b0010	As for 0b0001, and adds instructions for conversion between double-precision and half-precision.
0b0011	As for 0b0010, and adds support for half-precision floating-point arithmetic.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 in an implementation without floating-point support.
- 0b0010 in an implementation with floating-point support that does not include the FEAT_FP16 extension.
- 0b0011 in an implementation with floating-point support that includes the FEAT_FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the SIMDHP field, meaning the permitted values are:

Half-Precision instructions supported	FPHP	SIMDHP
No support	0b0000	0b0000
Conversions only	0b0010	0b0001
Conversions and arithmetic	0b0011	0b0010

Access to this field is **RO**.

SIMDHP, bits [23:20]

Advanced SIMD Half-Precision. Indicates the level of half-precision floating-point support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDHP	Meaning
0b0000	Not supported.
0b0001	SIMD half-precision conversion instructions are supported for conversion between single-precision and half-precision.
0b0010	As for 0b0001, and adds support for half-precision floating-point arithmetic.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 in an implementation without SIMD floating-point support.
- 0b0001 in an implementation with SIMD floating-point support that does not include the FEAT_FP16 extension.
- 0b0010 in an implementation with SIMD floating-point support that includes the FEAT_FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the FPHP field, meaning the permitted values are:

Half-Precision instructions supported	FPHP	SIMDHP
No support	0b0000	0b0000
Conversions only	0b0010	0b0001
Conversions and arithmetic	0b0011	0b0010

Access to this field is **RO**.

SIMDSP, bits [19:16]

Advanced SIMD Single-Precision. Indicates whether the Advanced SIMD and floating-point implementation provides single-precision floating-point instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDSP	Meaning
0b0000	Not implemented.
0b0001	Implemented. This value is permitted only if the SIMDInt field is 0b0001.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

Access to this field is **RO**.

SIMDInt, bits [15:12]

Advanced SIMD Integer. Indicates whether the Advanced SIMD and floating-point implementation provides integer instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDInt	Meaning
`0b0000`	Not implemented.
`0b0001`	Implemented.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

Access to this field is **RO**.

SIMDLS, bits [11:8]

Advanced SIMD Load/Store. Indicates whether the Advanced SIMD and floating-point implementation provides load/store instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDLS	Meaning
`0b0000`	Not implemented.
`0b0001`	Implemented.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

Access to this field is **RO**.

FPDNaN, bits [7:4]

Default NaN mode. Indicates whether the floating-point implementation provides support only for the Default NaN mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPDNaN	Meaning
`0b0000`	Not implemented, or hardware supports only the Default NaN mode.
`0b0001`	Hardware supports propagation of NaN values.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

Access to this field is **RO**.

FPFtZ, bits [3:0]

Flush to Zero mode. Indicates whether the floating-point implementation provides support only for the Flush-to-Zero mode of operation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPFtZ	Meaning
`0b0000`	Not implemented, or hardware supports only the Flush-to-Zero mode of operation.
`0b0001`	Hardware supports full denormalized number arithmetic.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

Access to this field is **RO**.

Accessing MVFR1

Accesses to this register use the following encodings in the System register encoding space:

```
VMRS{<c>}{<q>} <Rt>, <spec_reg>
```

reg
0b0110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = MVFR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS ==
'1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = MVFR1;
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        R[t] = MVFR1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR2, Media and VFP Feature Register 2

The MVFR2 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Must be interpreted with MVFR0 and MVFR1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register MVFR2 bits [31:0] are architecturally mapped to AArch64 System register MVFR2_EL1[31:0].

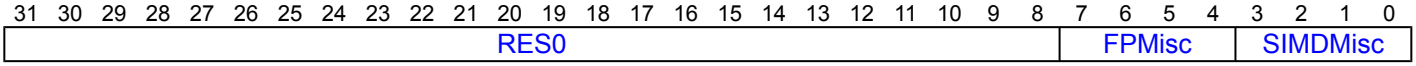
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to MVFR2 are UNDEFINED.

Implemented only if the implementation includes Advanced SIMD and floating-point instructions.

Attributes

MVFR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

FPMisc, bits [7:4]

Indicates whether the floating-point implementation provides support for miscellaneous VFP features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPMisc	Meaning
0b0000	Not implemented, or no support for miscellaneous features.
0b0001	Support for Floating-point selection.
0b0010	As 0b0001, and Floating-point Conversion to Integer with Directed Rounding modes.
0b0011	As 0b0010, and Floating-point Round to Integer Floating-point.
0b0100	As 0b0011, and Floating-point MaxNum and MinNum.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0100.

Access to this field is **RO**.

SIMDMisc, bits [3:0]

Indicates whether the Advanced SIMD implementation provides support for miscellaneous Advanced SIMD features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDMisc	Meaning
0b0000	Not implemented, or no support for miscellaneous features.
0b0001	Floating-point Conversion to Integer with Directed Rounding modes.
0b0010	As 0b0001, and Floating-point Round to Integer Floating-point.
0b0011	As 0b0010, and Floating-point MaxNum and MinNum.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0011.

Access to this field is **RO**.

Accessing MVFR2

Accesses to this register use the following encodings in the System register encoding space:

```
VMRS{<c>}{<q>} <Rt>, <spec_reg>
```

reg
0b0101

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
!ELIsInHost(EL2) && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif ELIsInHost(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
CPTR_EL2.FPEN IN {'x0'} then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) &&
((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0')
|| HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID3 ==
'1' then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = MVFR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ((IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SCR.NS ==
'1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
CPTR_EL3.TFP == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
    else
        R[t] = MVFR2;
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        R[t] = MVFR2;
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

NMRR, Normal Memory Remap Register

The NMRR characteristics are:

Purpose

Provides additional mapping controls for memory regions that are mapped as Normal memory by their entry in the PRRR.

Used in conjunction with the PRRR.

Configuration

This register is banked between NMRR and NMRR_S and NMRR_NS.

AArch32 System register NMRR bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL1[63:32] when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register NMRR bits [31:0] are architecturally mapped to AArch32 System register MAIR1[31:0] when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register NMRR bits [31:0] (NMRR_S) are architecturally mapped to AArch32 System register MAIR1[31:0] (MAIR1_S) when EL3 is using AArch32.

AArch32 System register NMRR bits [31:0] (NMRR_NS) are architecturally mapped to AArch32 System register MAIR1[31:0] (MAIR1_NS) when EL3 is using AArch32.

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to NMRR are UNDEFINED.

MAIR1 and NMRR are the same register, with a different view depending on the value of TTBCR.EAE:

- When it is set to 0, the register is as described in NMRR.
- When it is set to 1, the register is as described in MAIR1.

Attributes

NMRR is a 32-bit register.

This register has the following instances:

- NMRR, when EL3 is not implemented or FEAT_AA64 is implemented.
- NMRR_S, when FEAT_AA32EL3 is implemented.
- NMRR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When TTBCR.EAE == 0:

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
OR7	OR6	OR5	OR4	OR3	OR2	OR1	OR0	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0

OR<n>, bits [2n+17:2n+16], for n = 7 to 0

Outer Cacheable property mapping for memory attributes n, if the region is mapped as Normal memory by the PRRR.TR<n> entry. n is the value of the TEX[0], C, and B bits concatenated.

OR<n>	Meaning
0b00	Region is Non-cacheable.
0b01	Region is Write-Back, Write-Allocate.
0b10	Region is Write-Through, no Write-Allocate.
0b11	Region is Write-Back, no Write-Allocate.

The meaning of the field with n = 6 is IMPLEMENTATION DEFINED and might differ from the meaning given here. This is because the meaning of the attribute combination {TEX[0] = 1, C = 1, B = 0} is IMPLEMENTATION DEFINED.

When FEAT_XS is implemented, stage 1 Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IR<n>, bits [2n+1:2n], for n = 7 to 0

Inner Cacheable property mapping for memory attributes n, if the region is mapped as Normal memory by the PRRR.TR<n> entry. n is the value of the TEX[0], C, and B bits concatenated.

IR<n>	Meaning
0b00	Region is Non-cacheable.
0b01	Region is Write-Back, Write-Allocate.
0b10	Region is Write-Through, no Write-Allocate.
0b11	Region is Write-Back, no Write-Allocate.

The meaning of the field with n = 6 is IMPLEMENTATION DEFINED and might differ from the meaning given here. This is because the meaning of the attribute combination {TEX[0] = 1, C = 1, B = 0} is IMPLEMENTATION DEFINED.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing NMRR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            R[t] = MAIR1_NS;
        else
            R[t] = NMRR_NS;
    else
        if TTBCR.EAE == '1' then
            R[t] = MAIR1;
        else
            R[t] = NMRR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            R[t] = MAIR1_NS;
        else
            R[t] = NMRR_NS;
    else
        if TTBCR.EAE == '1' then
            R[t] = MAIR1;
        else
            R[t] = NMRR;
elsif PSTATE.EL == EL3 then
    if TTBCR.EAE == '1' then
        if SCR.NS == '0' then
            R[t] = MAIR1_S;
        else
            R[t] = MAIR1_NS;
    else
        if SCR.NS == '0' then
            R[t] = NMRR_S;
        else
            R[t] = NMRR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR1_NS = R[t];
        else
            NMRR_NS = R[t];
    else
        if TTBCR.EAE == '1' then
            MAIR1 = R[t];
        else
            NMRR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR1_NS = R[t];
        else
            NMRR_NS = R[t];
    else
        if TTBCR.EAE == '1' then
            MAIR1 = R[t];
        else
            NMRR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if TTBCR.EAE == '1' then
            if SCR.NS == '0' then
                MAIR1_S = R[t];
            else
                MAIR1_NS = R[t];
        else
            if SCR.NS == '0' then
                NMRR_S = R[t];
            else
                NMRR_NS = R[t];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

NSACR, Non-Secure Access Control Register

The NSACR characteristics are:

Purpose

When EL3 is implemented and can use AArch32, defines the Non-secure access permissions to Trace, Advanced SIMD and floating-point functionality. Also includes IMPLEMENTATION DEFINED bits that can define Non-secure access permissions for IMPLEMENTATION DEFINED functionality.

Configuration

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to NSACR are UNDEFINED.

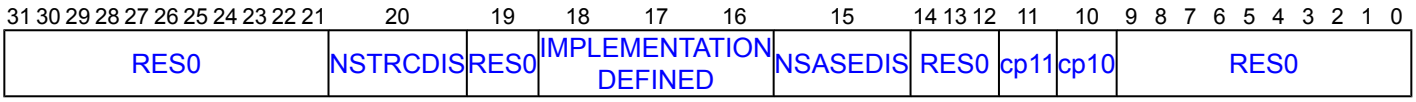
Note

In AArch64 state, the NSACR controls are replaced by controls in CPTR_EL3.

Attributes

NSACR is a 32-bit register.

Field descriptions



If EL3 is implemented and is using AArch64 then:

- Any read of the NSACR from Non-secure EL2 or Non-secure EL1 returns a value of `0x00000C00`.
- Any read or write to NSACR from Secure EL1 is trapped as an exception to EL3.

If EL3 is not implemented, then any read of the NSACR from EL2 or EL1 returns a value of `0x00000C00`.

Bits [31:21]

Reserved, RES0.

NSTRCDIS, bit [20]

Disables Non-secure System register accesses to all implemented trace registers.

NSTRCDIS	Meaning
`0b0`	This control has no effect on: • System register access to implemented trace registers. • The behavior of CPACR.TRCDIS and HCPTR.TTA.
`0b1`	Non-secure System register accesses to all implemented trace registers are disabled, meaning: • CPACR.TRCDIS behaves as RAO/WI in Non-secure state, regardless of its actual value. • HCPTR.TTA behaves as RAO/WI, regardless of its actual value.

The implementation of this field must correspond to the implementation of the CPACR.TRCDIS field:

- If CPACR.TRCDIS is RAZ/WI, this field is RAZ/WI.
- If CPACR.TRCDIS is RW, this field is RW.

Note

- The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED.
- The Arm architecture does not provide Non-secure access controls on trace register accesses through the optional memory-mapped external debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to `'0'`.

Bit [19]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [18:16]

IMPLEMENTATION DEFINED.

NSASEDIS, bit [15]

Disables Non-secure access to the Advanced SIMD functionality.

NSASEDIS	Meaning
`0b0`	This control has no effect on: • Non-secure access to Advanced SIMD functionality. • The behavior of CPACR.ASEDIS and HCPTR.TASE.
`0b1`	Non-secure access to the Advanced SIMD functionality is disabled, meaning: • CPACR.ASEDIS behaves as RAO/WI in Non-secure state, regardless of its actual value. • HCPTR.TASE behaves as RAO/WI, regardless of its actual value.

The implementation of this field must correspond to the implementation of the CPACR.ASEDIS field:

- If CPACR.ASEDIS is RES0, this field is RES0. If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.
- If CPACR.ASEDIS is RAZ/WI, this field is RAZ/WI.
- If CPACR.ASEDIS is RW, this field is RW.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to `'0'`.

Bits [14:12]

Reserved, RES0.

cp11, bit [11]

The value of this field is ignored. If this field is programmed with a different value to the cp10 field then this field is UNKNOWN on a direct read of the NSACR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

cp10, bit [10]

Enable Non-secure access to the Advanced SIMD and floating-point features. Possible values of the fields are:

cp10	Meaning
0b0	Advanced SIMD and floating-point features can be accessed only from Secure state. Any attempt to access this functionality from Non-secure state is UNDEFINED. When the PE is in Non-secure state: • The CPACR.{cp11, cp10} fields ignore writes and read as 0b00, access denied. • The HCPTR.{TCP11, TCP10} fields behave as RAO/WI, regardless of their actual values.
0b1	Advanced SIMD and floating-point features can be accessed from both Security states.

If Non-secure access to the Advanced SIMD and floating-point functionality is enabled, the CPACR must be checked to determine the level of access that is permitted.

The Advanced SIMD and floating-point features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the FPSCR, FPSID, MVFR0, MVFR1, MVFR2, or FPEXC System registers.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing NSACR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif !HaveEL(EL3) || (IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
SCR_EL3.NS == '1') then
        R[t] = Zeros(20):'1100':Zeros(8);
    else
        R[t] = NSACR;
elsif PSTATE.EL == EL2 then
    if !HaveEL(EL3) || (IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS
== '1') then
        R[t] = Zeros(20):'1100':Zeros(8);
    else
        R[t] = NSACR;
elsif PSTATE.EL == EL3 then
    R[t] = NSACR;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        NSACR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PAR, Physical Address Register

The PAR characteristics are:

Purpose

Returns the output address (OA) from an Address translation instruction that executed successfully, or fault information if the instruction did not execute successfully.

Configuration

This register is banked between PAR and PAR_S and PAR_NS.

AArch32 System register PAR bits [63:0] are architecturally mapped to AArch64 System register PAR_EL1[63:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to PAR are UNDEFINED.

PAR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits[31:0] and do not modify bits[63:32].

The Configurations section specifies the cases where each PAR format is used.

PAR is accessed as a 32-bit value:

- When the PE is not in Hyp mode and is using the Short-descriptor translation table format.
- When the PE is in Hyp mode and executes an ATS12NSOPR, ATS12NSOPW, ATS12NSOUR, or ATS12NSOUW instruction and the value of HCR.VM is 0 and the value of TTBCR.EAE is 0.

In these cases, PAR[63:32] is RES0.

Otherwise, the PAR is accessed as a 64-bit value, if any of the following is true:

- When using the Long-descriptor translation table format.
- If the stage 1 address translation is disabled and TTBCR.EAE is set to 1.
- In an implementation that includes EL2, for the result of an ATS1Cxx instruction performed from Hyp mode.

For PL1&0 stage 1 translations, TTBCR.EAE selects the translation table format.

Attributes

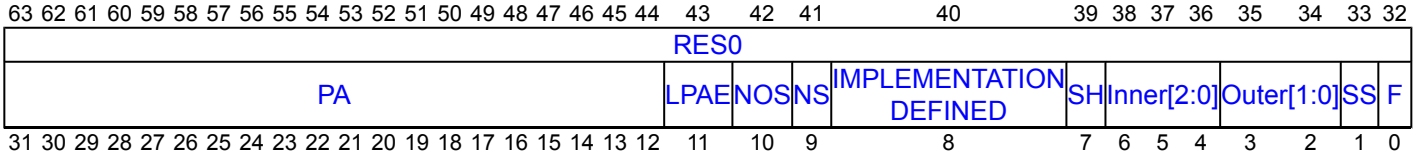
PAR is a 64-bit register.

This register has the following instances:

- PAR, when EL3 is not implemented or FEAT_AA64 is implemented.
- PAR_S, when FEAT_AA32EL3 is implemented.
- PAR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When the instruction returned a 32-bit value to the PAR, PAR.F==0:



This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR can return a value that indicates the resulting attributes, rather than the values that appear in the Translation table descriptors. More precisely:

- Memory attribute fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the Translation table descriptors. This applies to the NOS, SH, Inner, and Outer fields.
- See the NS bit description for constraints on the value it returns.

Bits [63:32]

Reserved, RES0.

PA, bits [31:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[31:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [11]

When updating the PAR with the result of the translation operation, this bit is set as follows:

LPAE	Meaning
0b0	Short-descriptor translation table format used. This means the PAR returned a 32-bit value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NOS, bit [10]

Not Outer Shareable. When the returned value of PAR.SH is 1, indicates the Shareability attribute for the physical memory region:

NOS	Meaning
0b0	Memory region is Outer Shareable.
0b1	Memory region is Inner Shareable.

When the returned value of PAR.SH is 0 the value returned to this field is UNKNOWN.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bit [8]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bit [7]

Shareability. Indicates whether the physical memory region is Non-shareable:

SH	Meaning
0b0	Memory is Non-shareable.
0b1	Memory is shareable, and PAR.NOS indicates whether the region is Outer Shareable or Inner Shareable.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Inner[2:0], bits [6:4]

Inner cacheability attribute for the region. Permitted values are:

Inner[2:0]	Meaning
0b000	Non-cacheable.
0b001	Device-nGnRnE.
0b011	Device-nGnRE.
0b101	Write-Back, Write-Allocate.
0b110	Write-Through.
0b111	Write-Back, no Write-Allocate.

The values 0b010 and 0b100 are reserved.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Outer[1:0], bits [3:2]

Outer cacheability attribute for the region. Permitted values are:

Outer[1:0]	Meaning
0b00	Non-cacheable.
0b01	Write-Back, Write-Allocate.
0b10	Write-Through, no Write-Allocate.
0b11	Write-Back, no Write-Allocate.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SS, bit [1]

Supersection. Used to indicate if the result is a Supersection:

SS	Meaning
0b0	Result is not a Supersection. PAR[31:12] contains OA[31:12].
0b1	Result is a Supersection, and: • PAR[31:24] contains OA[31:24]. • PAR[23:16] contains OA[39:32]. • PAR[15:12] contains 0b0000. If an implementation supports less than 40 bits of physical address, the bits in the PAR field that correspond to physical address bits that are not implemented are UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

F	Meaning
0b0	Address translation completed successfully.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When the instruction returned a 32-bit value to the PAR, PAR.F==1:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
IMPLEMENTATION DEFINED																RES0				LPAE	RES0				FS[5]	FS[4:0]					F
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:16]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:12]

Reserved, RES0.

LPAE, bit [11]

When updating the PAR with the result of the translation operation, this bit is set as follows:

LPAE	Meaning
0b0	Short-descriptor translation table format used. This means the PAR returned a 32-bit value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:7]

Reserved, RES0.

FS[5], bit [6]

Fault status bits, External abort type. Provides an IMPLEMENTATION DEFINED classification of an External abort. Values are as in the DFSR.ExT field when using the Short-descriptor translation table format.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FS[4:0], bits [5:1]

Fault status bits. Values are as in the DFSR.FS field when using the Short-descriptor translation table format.

FS[4:0]	Meaning	Applies when
0b00001	Alignment fault.	
0b00011	Access flag fault, level 1.	
0b00100	Fault on instruction cache maintenance.	
0b00101	Translation fault, level 1.	
0b00110	Access flag fault, level 2.	
0b00111	Translation fault, level 2.	
0b01001	Domain fault, level 1.	
0b01011	Domain fault, level 2.	
0b01100	Synchronous External abort, on translation table walk, level 1.	
0b01101	Permission fault, level 1.	
0b01110	Synchronous External abort, on translation table walk, level 2.	
0b01111	Permission fault, level 2.	
0b10000	TLB conflict abort.	
0b11001	Synchronous parity or ECC error on memory access, not on translation table walk.	When FEAT_RAS is not implemented
0b11100	Synchronous parity or ECC error on translation table walk, level 1.	When FEAT_RAS is not implemented
0b11110	Synchronous parity or ECC error on translation table walk, level 2.	When FEAT_RAS is not implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

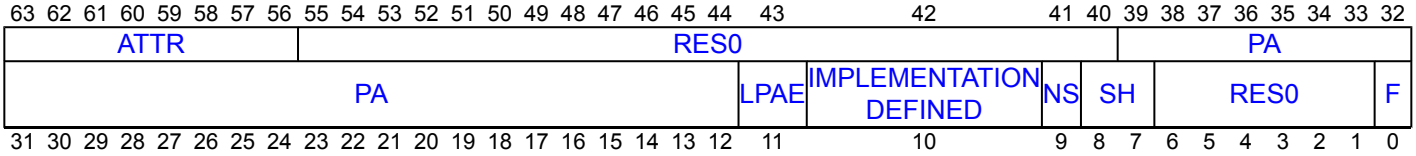
Indicates whether the instruction performed a successful address translation.

F	Meaning
0b1	Address translation aborted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When the instruction returned a 64-bit value to the PAR, PAR.F==0:



This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR can return a value that indicates the resulting attributes, rather than the values that appear in the Translation table descriptors. More precisely:

- Memory attribute fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the Translation table descriptors. This applies to the ATTR and SH fields.
- See the NS bit description for constraints on the value it returns.

ATTR, bits [63:56]

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in MAIR0 and MAIR1.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [55:40]

Reserved, RES0.

PA, bits [39:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[39:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [11]

When updating the PAR with the result of the translation operation, this bit is set as follows:

LPAE	Meaning
0b1	Long-descriptor translation table format used. This means the PAR returned a 64-bit value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bit [10]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [8:7]

Shareability attribute, for the returned output address. Permitted values are:

SH	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

The value 0b01 is reserved.

Note

This field returns the value 0b10 for:

- Any type of Device memory.
- Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:1]

Reserved, RES0.

F, bit [0]

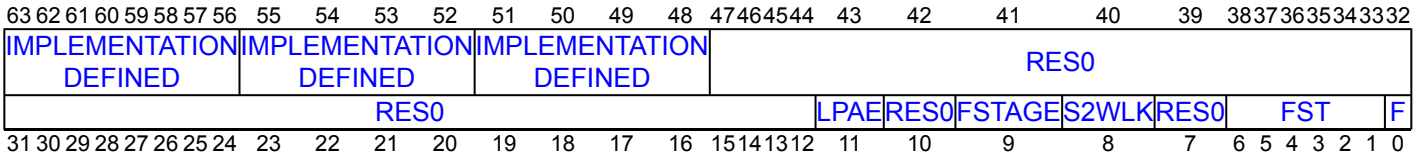
Indicates whether the instruction performed a successful address translation.

F	Meaning
0b0	Address translation completed successfully.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When the instruction returned a 64-bit value to the PAR, PAR.F==1:



This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

IMPLEMENTATION DEFINED, bits [63:56]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [55:52]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [51:48]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:12]

Reserved, RES0.

LPAE, bit [11]

When updating the PAR with the result of the translation operation, this bit is set as follows:

LPAE	Meaning
0b1	Long-descriptor translation table format used. This means the PAR returned a 64-bit value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [10]

Reserved, RES0.

FSTAGE, bit [9]

Indicates the translation stage at which the translation aborted:

FSTAGE	Meaning
0b0	Translation aborted because of a fault in the stage 1 translation.
0b1	Translation aborted because of a fault in the stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S2WLK, bit [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

FST, bits [6:1]

Fault status field. Values are as in the DFSR.STATUS and IFSR.STATUS fields when using the Long-descriptor translation table format.

FST	Meaning	Applies when
0b000000	Address size fault in translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010101	Synchronous External abort on translation table walk, level 1.	
0b010110	Synchronous External abort on translation table walk, level 2.	
0b010111	Synchronous External abort on translation table walk, level 3.	
0b011101	Synchronous parity or ECC error on memory access on translation table walk, level 1.	When FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk, level 2.	When FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk, level 3.	When FEAT_RAS is not implemented
0b110000	TLB conflict abort.	

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

F	Meaning
0b1	Address translation aborted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = PAR_NS<31:0>;
    else
        R[t] = PAR<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = PAR_NS<31:0>;
    else
        R[t] = PAR<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = PAR_S<31:0>;
    else
        R[t] = PAR_NS<31:0>;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0111	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        PAR_NS<31:0> = R[t];
    else
        PAR<31:0> = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        PAR_NS<31:0> = R[t];
    else
        PAR<31:0> = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        PAR_S<31:0> = R[t];
    else
        PAR_NS<31:0> = R[t];
```

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

coproc	CRm	opc1
0b1111	0b0111	0b0000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = PAR_NS;
    else
        R[t, t2] = PAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = PAR_NS;
    else
        R[t, t2] = PAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t, t2] = PAR_S;
    else
        R[t, t2] = PAR_NS;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b0111	0b0000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T7
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T7 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        PAR_NS = R[t, t2];
    else
        PAR = R[t, t2];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        PAR_NS = R[t, t2];
    else
        PAR = R[t, t2];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        PAR_S = R[t, t2];
    else
        PAR_NS = R[t, t2];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCFILTR, Performance Monitors Cycle Count Filter Register

The PMCCFILTR characteristics are:

Purpose

Determines the modes in which the Cycle Counter, PMCCNTR, increments.

Configuration

AArch32 System register PMCCFILTR bits [31:0] are architecturally mapped to AArch64 System register PMCCFILTR_EL0[31:0].

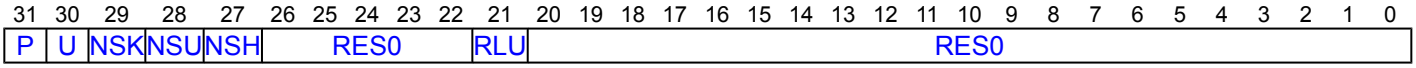
AArch32 System register PMCCFILTR bits [31:0] are architecturally mapped to External register PMCCFILTR_EL0[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCCFILTR are UNDEFINED.

Attributes

PMCCFILTR is a 32-bit register.

Field descriptions



P, bit [31]

Privileged filtering. Controls counting cycles in EL1 and, if EL3 is using AArch32, EL3.

P	Meaning
0b0	This mechanism has no effect on filtering of cycles.
0b1	The PE does not count cycles in EL1 and, if EL3 is using AArch32, EL3.

If Secure and Non-secure states are implemented, then counting cycles in Non-secure EL1 is further controlled by PMCCFILTR.NSK.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

User filtering. Controls counting cycles in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of cycles.
0b1	The PE does not count cycles in EL0.

If Secure and Non-secure states are implemented, then counting cycles in Non-secure EL0 is further controlled by PMCCFILTR.NSU.

If FEAT_RME is implemented, then counting cycles in Realm EL0 is further controlled by PMCCFILTR.RLU.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:

- When FEAT_AA64 is not implemented, this field resets to '0'.
- When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting cycles in Non-secure EL1. If PMCCFILTR.NSK is not equal to PMCCFILTR.P, then the PE does not count cycles in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of cycles in Non-secure EL1.

NSK	Meaning
0b0	When PMCCFILTR.P == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR.P == 1, the PE does not count cycles in Non-secure EL1.
0b1	When PMCCFILTR.P == 0, the PE does not count cycles in Non-secure EL1. When PMCCFILTR.P == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 filtering. Controls counting cycles in Non-secure EL0. If PMCCFILTR.NSU is not equal to PMCCFILTR.U, then the PE does not count cycles in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of cycles in Non-secure EL0.

NSU	Meaning
0b0	When PMCCFILTR.U == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR.U == 1, the PE does not count cycles in Non-secure EL0.
0b1	When PMCCFILTR.U == 0, the PE does not count cycles in Non-secure EL0. When PMCCFILTR.U == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 filtering. Controls counting cycles in EL2.

NSH	Meaning
0b0	The PE does not count cycles in EL2.
0b1	This mechanism has no effect on filtering of cycles.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting cycles in Secure EL2 is further controlled by PMCCFILTR.SH.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to `'0'`.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [26:22]

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting cycles in Realm EL0. If PMCCFILTR.RLU is not equal to PMCCFILTR.U, then the PE does not count cycles in Realm EL0. Otherwise, this mechanism has no effect on filtering of cycles in Realm EL0.

RLU	Meaning
`0b0`	When PMCCFILTR.U == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR.U == 1, the PE does not count cycles in Realm EL0.
`0b1`	When PMCCFILTR.U == 0, the PE does not count cycles in Realm EL0. When PMCCFILTR.U == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [20:0]

Reserved, RES0.

Accessing PMCCFILTR

PMCCFILTR can also be accessed by using PMXEVTYPER with PMSELR.SEL set to `0b11111`.

Permitted reads and writes of PMCCFILTR are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.C == 0.

Permitted writes of PMCCFILTR are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.{UEN,CR} == {1,1}.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b1111	0b111

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.C == '0' then
            R[t] = Zeros(32);
        else
            R[t] = PMCCFILTR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCCFILTR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```

```
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCCFILTR;
elsif PSTATE.EL == EL3 then
    R[t] = PMCCFILTR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b1111	0b111

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.C == '0' || PMUSERENR_EL0.CR ==
'1') then
            return;
        else
            PMCCFILTR = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCFILTR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCFILTR = R[t];
elsif PSTATE.EL == EL3 then
    PMCCFILTR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCNTR, Performance Monitors Cycle Count Register

The PMCCNTR characteristics are:

Purpose

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' for more information.

PMCCFILTR determines the modes and states in which the PMCCNTR can increment.

Configuration

AArch32 System register PMCCNTR bits [63:0] are architecturally mapped to AArch64 System register PMCCNTR_EL0[63:0].

AArch32 System register PMCCNTR bits [63:0] are architecturally mapped to External register PMCCNTR_EL0[63:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCCNTR are UNDEFINED.

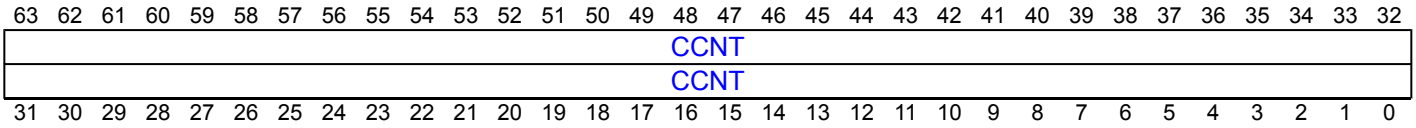
PMCCNTR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

All counters are subject to any changes in clock frequency, including clock stopping caused by the WFI and WFE instructions. This means that it is CONSTRAINED UNPREDICTABLE whether or not PMCCNTR continues to increment when clocks are stopped by WFI and WFE instructions.

Attributes

PMCCNTR is a 64-bit register.

Field descriptions



CCNT, bits [63:0]

Cycle count. Depending on the values of PMCR.{LC,D}, this field increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to PMCR.C sets this field to 0.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMCCNTR

Permitted reads and writes of PMCCNTR are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.UEN == 1.

- PMUACR_EL1.C == 0.

Permitted writes of PMCCNTR are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.{UEN,CR} == {1,1}.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1101	0b000

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
((IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,CR,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<CR,EN> == '00')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.<CR,EN> == '00'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.C == '0' then
            R[t] = Zeros(32);
        else
            R[t] = PMCCNTR<31:0>;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCCNTR<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCCNTR<31:0>;
elsif PSTATE.EL == EL3 then
    R[t] = PMCCNTR<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1101	0b000

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.C == '0' || PMUSERENR_EL0.CR ==
'1') then
            return;
        else
            PMCCNTR<31:0> = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCNTR<31:0> = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCNTR<31:0> = R[t];
elsif PSTATE.EL == EL3 then
    PMCCNTR<31:0> = R[t];
```

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1001	0b0000

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
((IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,CR,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<CR,EN> == '00')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.<CR,EN> == '00'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.C == '0' then
            R[t, t2] = Zeros(64);
        else
            R[t, t2] = PMCCNTR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
```

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        R[t, t2] = PMCCNTR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        R[t, t2] = PMCCNTR;
elsif PSTATE.EL == EL3 then
    R[t, t2] = PMCCNTR;
```

```
MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b1001	0b0000

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.C == '0' || PMUSERENR_EL0.CR ==
'1') then
            return;
        else
            PMCCNTR = R[t, t2];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
```

```
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t, t2];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t, t2];
elsif PSTATE.EL == EL3 then
    PMCCNTR = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID0, Performance Monitors Common Event Identification register 0

The PMCEID0 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range `0x0000` to `0x001F`.

For more information about the Common events and the use of the PMCEIDn registers, see 'The PMU event number space and common events'.

Configuration

AArch32 System register PMCEID0 bits [31:0] are architecturally mapped to AArch64 System register PMCEID0_EL0[31:0].

AArch32 System register PMCEID0 bits [31:0] are architecturally mapped to External register PMCEID0[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCEID0 are UNDEFINED.

Attributes

PMCEID0 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4
ID31	ID30	ID29	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event n.

For each bit:

ID<n>	Meaning
`0b0`	The Common event is not implemented, or not counted.
`0b1`	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.TID == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) &&
IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR.TID == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
```

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID0;
elsif PSTATE.EL == EL3 then
    R[t] = PMCEID0;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID1, Performance Monitors Common Event Identification register 1

The PMCEID1 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range `0x0020` to `0x003F`.

For more information about the Common events and the use of the PMCEIDn registers see 'The PMU event number space and common events'.

Configuration

AArch32 System register PMCEID1 bits [31:0] are architecturally mapped to AArch64 System register PMCEID1_EL0[31:0].

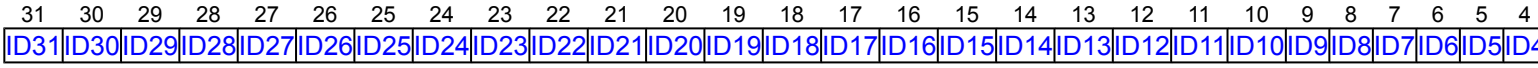
AArch32 System register PMCEID1 bits [31:0] are architecturally mapped to External register PMCEID1[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCEID1 are UNDEFINED.

Attributes

PMCEID1 is a 32-bit register.

Field descriptions



ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event (`0x0020` + n).

For each bit:

ID<n>	Meaning
`0b0`	The Common event is not implemented, or not counted.
`0b1`	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.TID == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) &&
IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR.TID == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID1;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
```

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID1;
elsif PSTATE.EL == EL3 then
    R[t] = PMCEID1;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID2, Performance Monitors Common Event Identification register 2

The PMCEID2 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range `0x4000` to `0x401F`.

For more information about the Common events and the use of the PMCEIDn registers see 'The PMU event number space and common events'.

Configuration

AArch32 System register PMCEID2 bits [31:0] are architecturally mapped to AArch64 System register PMCEID0_EL0[63:32].

AArch32 System register PMCEID2 bits [31:0] are architecturally mapped to External register PMCEID2[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3p1 is implemented. Otherwise, direct accesses to PMCEID2 are UNDEFINED.

Attributes

PMCEID2 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	
IDhi31	IDhi30	IDhi29	IDhi28	IDhi27	IDhi26	IDhi25	IDhi24	IDhi23	IDhi22	IDhi21	IDhi20	IDhi19	IDhi18	IDhi17	IDhi16	IDhi15	IDhi14	IDhi13	IDh

IDhi<n>, bit [n], for n = 31 to 0

IDhi[n] corresponds to Common event (`0x4000` + n).

For each bit:

IDhi<n>	Meaning
`0b0`	The Common event is not implemented, or not counted.
`0b1`	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID2

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b100

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3p1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.TID == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) &&
IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR.TID == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID2;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
```

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID2;
elsif PSTATE.EL == EL3 then
    R[t] = PMCEID2;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID3, Performance Monitors Common Event Identification register 3

The PMCEID3 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range `0x4020` to `0x403F`.

For more information about the Common events and the use of the PMCEIDn registers see 'The PMU event number space and common events'.

Configuration

AArch32 System register PMCEID3 bits [31:0] are architecturally mapped to AArch64 System register PMCEID1_EL0[63:32].

AArch32 System register PMCEID3 bits [31:0] are architecturally mapped to External register PMCEID3[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3p1 is implemented. Otherwise, direct accesses to PMCEID3 are UNDEFINED.

Attributes

PMCEID3 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	
IDhi31	IDhi30	IDhi29	IDhi28	IDhi27	IDhi26	IDhi25	IDhi24	IDhi23	IDhi22	IDhi21	IDhi20	IDhi19	IDhi18	IDhi17	IDhi16	IDhi15	IDhi14	IDhi13	IDh

IDhi<n>, bit [n], for n = 31 to 0

IDhi[n] corresponds to Common event (`0x4020` + n).

For each bit:

IDhi<n>	Meaning
`0b0`	The Common event is not implemented, or not counted.
`0b1`	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID3

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b101

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3p1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.TID == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) &&
IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR.TID == '1' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID3;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
```

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID3;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCEID3;
elsif PSTATE.EL == EL3 then
    R[t] = PMCEID3;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTENCLR, Performance Monitors Count Enable Clear register

The PMCNTENCLR characteristics are:

Purpose

Allows software to disable the following counters:

- The cycle counter PMCCNTR.
- The event counters PMEVCNTR<n>.

Reading from this register shows which counters are enabled.

Configuration

AArch32 System register PMCNTENCLR bits [31:0] are architecturally mapped to AArch32 System register PMCNTENSET[31:0].

AArch32 System register PMCNTENCLR bits [31:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[31:0].

AArch32 System register PMCNTENCLR bits [31:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[31:0].

AArch32 System register PMCNTENCLR bits [31:0] are architecturally mapped to External register PMCNTENCLR_EL0[31:0].

AArch32 System register PMCNTENCLR bits [31:0] are architecturally mapped to External register PMCNTENSET_EL0[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCNTENCLR are UNDEFINED.

Attributes

PMCNTENCLR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

C, bit [31]

PMCCNTR disable. On writes, allows software to disable PMCCNTR. On reads, returns the PMCCNTR enable status.

C	Meaning
0b0	PMCCNTR disabled.
0b1	PMCCNTR enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - EL1 is using AArch64.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.C == 0.

- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - EL1 is using AArch64.
  - PMUSERENR_EL0.[UEN,CR] == 0b11.
- Otherwise, access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

PMEVCNTR<m> disable. On writes, allows software to disable PMEVCNTR<m>. On reads, returns the PMEVCNTR<m> enable status.

P<m>	Meaning
0b0	PMEVCNTR<m> disabled.
0b1	PMEVCNTR<m> enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - EL1 is using AArch64.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.P<m> == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - EL1 is using AArch64.
  - PMUSERENR_EL0.[UEN,ER] == 0b11.
- Otherwise, access to this field is **W1C**.

Accessing PMCNTENCLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCNTENCLR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCNTENCLR;
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCNTENCLR;
elsif PSTATE.EL == EL3 then
    R[t] = PMCNTENCLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENCLR = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENCLR = R[t];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENCLR = R[t];
elsif PSTATE.EL == EL3 then
    PMCNTENCLR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTENSET, Performance Monitors Count Enable Set register

The PMCNTENSET characteristics are:

Purpose

Allows software to enable the following counters:

- The cycle counter PMCCNTR.
- The event counters PMEVCNTR<n>.

Reading from this register shows which counters are enabled.

Configuration

AArch32 System register PMCNTENSET bits [31:0] are architecturally mapped to AArch32 System register PMCNTENCLR[31:0].

AArch32 System register PMCNTENSET bits [31:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[31:0].

AArch32 System register PMCNTENSET bits [31:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[31:0].

AArch32 System register PMCNTENSET bits [31:0] are architecturally mapped to External register PMCNTENSET_EL0[31:0].

AArch32 System register PMCNTENSET bits [31:0] are architecturally mapped to External register PMCNTENCLR_EL0[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCNTENSET are UNDEFINED.

Attributes

PMCNTENSET is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

C, bit [31]

PMCCNTR enable. On writes, allows software to enable PMCCNTR. On reads, returns the PMCCNTR enable status.

C	Meaning
0b0	PMCCNTR disabled.
0b1	PMCCNTR enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - EL1 is using AArch64.
  - PMUSERENR_EL0.UEN == 1.
  - PMUACR_EL1.C == 0.
- Access to this field is **RO** if all the following are true:
  - FEAT_PMUv3p9 is implemented.

        - PSTATE.EL == EL0.
        - EL1 is using AArch64.
        - PMUSERENR_EL0.[UEN,CR] == 0b11.
    - Otherwise, access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

PMEVCNTR<m> enable. On writes, allows software to enable PMEVCNTR<m>. On reads, returns the PMEVCNTR<m> enable status.

P<m>	Meaning
0b0	PMEVCNTR<m> disabled.
0b1	PMEVCNTR<m> enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
    - FEAT_PMUv3p9 is implemented.
    - PSTATE.EL == EL0.
    - EL1 is using AArch64.
    - PMUSERENR_EL0.UEN == 1.
    - PMUACR_EL1.P<m> == 0.
- Access to this field is **RO** if all the following are true:
    - FEAT_PMUv3p9 is implemented.
    - PSTATE.EL == EL0.
    - EL1 is using AArch64.
    - PMUSERENR_EL0.[UEN,ER] == 0b11.
- Otherwise, access to this field is **W1S**.

Accessing PMCNTENSET

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCNTENSET;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCNTENSET;
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCNTENSET;
elsif PSTATE.EL == EL3 then
    R[t] = PMCNTENSET;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENSET = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENSET = R[t];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENSET = R[t];
elsif PSTATE.EL == EL3 then
    PMCNTENSET = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCR, Performance Monitors Control Register

The PMCR characteristics are:

Purpose

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Configuration

AArch32 System register PMCR bits [31:0] are architecturally mapped to AArch64 System register PMCR_EL0[31:0].

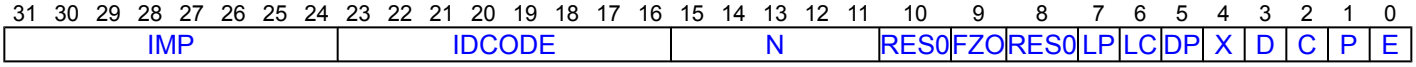
AArch32 System register PMCR bits [10:0] are architecturally mapped to External register PMCR_EL0[10:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCR are UNDEFINED.

Attributes

PMCR is a 32-bit register.

Field descriptions



IMP, bits [31:24]

When FEAT_PMUv3p7 is not implemented:

Implementer code.

If this field is zero, then PMCR.IDCODE is RES0 and software must use MIDR to identify the PE.

Otherwise, this field and PMCR.IDCODE identify the PMU implementation to software. The implementer codes are allocated by Arm. A nonzero value has the same interpretation as MIDR.Implementer.

Arm deprecates use of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

IDCODE, bits [23:16]

When PMCR.IMP != 0b00000000:

Identification code. Arm deprecates use of this field.

Each implementer must maintain a list of identification codes that are specific to the implementer. A specific implementation is identified by the combination of the implementer code and the identification code.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

N, bits [15:11]

Indicates the number of event counters implemented. This value is in the range of `0b00000`-`0b11111`. If the value is `0b00000`, then only PMCCNTR is implemented. If the value is `0b11111`, then PMCCNTR and 31 event counters are implemented.

If EL2 is implemented, then all of the following apply:

- If EL2 is using AArch32, then reads of this field from Non-secure EL1 and Non-secure EL0 return the Effective value of HDCR.HPMN.
- If EL2 is enabled in the current Security state and using AArch64, then reads of this field from EL1 and EL0 return the Effective value of MDCR_EL2.HPMN.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [10]

Reserved, RES0.

FZO, bit [9]

When FEAT_PMUv3p7 is implemented:

Freeze-on-overflow. Stop event counters on overflow.

FZO	Meaning
`0b0`	Do not freeze on overflow.
`0b1`	Affected counters do not count when PMOVSR[m] is 1 for any event counter PMEVCNTR<m> in the first range.

The counters affected by this field are:

- The event counters in the first range.
- If PMCR.DP is 1, the cycle counter PMCCNTR.

Other event counters are not affected by this field.

When PMCR.DP is 0, PMCCNTR is not affected by this field.

For more information about event counter ranges, see MDCR_EL2.HPMN or HDCR.HPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

LP, bit [7]

When FEAT_PMUv3p5 is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by PMOVSR.P[n].

LP	Meaning
0b0	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>[31:0].
0b1	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>[63:0].

The counters affected by this field are the event counters in the first range. For more information about event counter ranges, see MDCR_EL2.HPMN or HDCR.HPMN.

Other event counters and PMCCNTR are not affected by this field.

PMEVCNTR<n>[63:32] is not accessible in AArch32 state.

If the highest implemented Exception level is using AArch32, it is IMPLEMENTATION DEFINED whether this field is read/write or RAZ/WI.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LC, bit [6]

Long cycle counter enable. Determines when unsigned overflow is recorded by PMOVSR.C.

LC	Meaning
0b0	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR[31:0].
0b1	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR[63:0].

Arm deprecates use of PMCR.LC = 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DP, bit [5]

When EL3 is implemented or (FEAT_PMUv3p1 is implemented and EL2 is implemented):

Disable cycle counter when event counting is prohibited.

DP	Meaning
0b0	Cycle counting by PMCCNTR is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR is disabled in prohibited regions and when event counting is frozen: • If FEAT_PMUv3p1 is implemented, EL2 is implemented, and MDCR_EL2.HPMD or HDCR.HPMD is 1, then cycle counting by PMCCNTR is disabled at EL2. • If FEAT_PMUv3p7 is implemented and event counting is frozen by PMCR.FZO, then cycle counting by PMCCNTR is disabled. • If FEAT_PMUv3p7 is implemented, EL3 is implemented and using AArch64, and MDCR_EL3.MPMX is 1, then cycle counting by PMCCNTR is disabled at EL3. • If EL3 is implemented, MDCR_EL3.SPME or SDCR.SPME is 0, and one of FEAT_PMUv3p7 is not implemented, EL3 is using AArch32, or MDCR_EL3.MPMX is 0, then cycle counting by PMCCNTR is disabled at EL3 and in Secure state.

The conditions when this field disables the cycle counter are the same as when event counting by an event counter in the first range is prohibited or frozen. For more information about event counter ranges, see MDCR_EL2.HPMN or HDCR.HPMN.

For more information, see 'Prohibiting event and cycle counting'.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

X, bit [4]
When the implementation includes a PMU event export bus:

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.

X	Meaning
0b0	Do not export events.
0b1	Export events where not prohibited.

This field enables the exporting of events over an IMPLEMENTATION DEFINED PMU event export bus to another device.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

If FEAT_ETE is implemented, this field does not affect the use of PMU events as an External Input by the trace unit.

If FEAT_ETMv4 is implemented, this field does affect the use of PMU events as an External Input by the trace unit.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

D, bit [3]

Clock divider.

D	Meaning
0b0	When enabled, PMCCNTR counts every clock cycle.
0b1	When enabled, PMCCNTR counts once every 64 clock cycles.

If the Effective value of PMCR.LC is 1, then this field is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR.D = 1.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Cycle counter reset. The effects of writing to this field are:

C	Meaning
0b0	No action.
0b1	Reset PMCCNTR to zero.

Note

Resetting PMCCNTR does not change the cycle counter overflow field. The value of PMCR.LC is ignored, and bits [63:0] of the cycle counter are reset.

Access to this field is **WO/RAZ**.

P, bit [1]

Event counter reset.

P	Meaning
0b0	No action.
0b1	Reset all affected event counters PMEVCNTR<n> to zero.

The event counters affected by this field are:

- All event counters in the first range.
- If any of the following are true, all event counters in the second range:
  - EL2 is disabled or not implemented in the current Security state.
  - The PE is executing at EL2 or EL3.

Writes to this field do not affect other event counters or the cycle counter PMCCNTR.

For more information about event counter ranges, see MDCR_EL2.HPMN or HDCR.HPMN.

Note

Resetting the event counters does not change the event counter overflow fields. If FEAT_PMUv3p5 is implemented, the values of MDCR_EL2.HLP or HDCR.HLP and PMCR.LP are ignored, and bits [63:0] of all affected event counters are reset.

Access to this field is **WO/RAZ**.

E, bit [0]

Enable.

E	Meaning
0b0	Affected counters are disabled and do not count.
0b1	Affected counters are enabled by PMCNTENSET.

The counters affected by this field are:

- The event counters in the first range. For more information about event counter ranges, see MDCR_EL2.HPMN or HDCR.HPMN.
- The cycle counter PMCCNTR.

Other event counters are not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing PMCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
|| (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HDCR.TPMCR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPMCR
== '1' then
        AArch32.TakeHypTrapException(0x03);
```

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMCR;
elsif PSTATE.EL == EL3 then
    R[t] = PMCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
|| (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HDCR.TPMCR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCR = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
```

```
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPMCR
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCR = R[t];
elsif PSTATE.EL == EL3 then
    PMCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVCNTR<n>, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n> characteristics are:

Purpose

Holds event counter n, which counts events, where n is 0 to 30.

Configuration

AArch32 System register PMEVCNTR<n> bits [31:0] are architecturally mapped to AArch64 System register PMEVCNTR<n>_EL0[31:0].

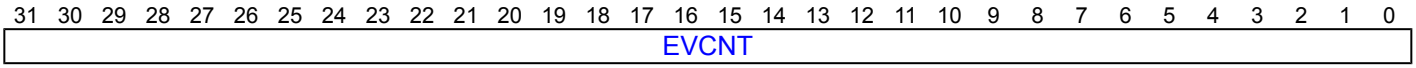
AArch32 System register PMEVCNTR<n> bits [31:0] are architecturally mapped to External register PMEVCNTR<n>_EL0[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMEVCNTR<n> are UNDEFINED.

Attributes

PMEVCNTR<n> is a 32-bit register.

Field descriptions



EVCNT, bits [31:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

If FEAT_PMUv3p5 is implemented, the event counter is 64 bits and only the least-significant part of the event counter is accessible in AArch32 state:

- Reads from PMEVCNTR<n> return bits [31:0] of the counter.
- Writes to PMEVCNTR<n> update bits [31:0] and leave bits [63:32] unchanged.
- There is no means to access bits [63:32] directly from AArch32 state.
- If the implementation does not support AArch64, bits [63:32] are not required to be implemented.

If FEAT_PMUv3p5 is not implemented, the event counter is 32 bits.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMEVCNTR<n>

PMEVCNTR<n> can also be accessed by using PMXEVCNTR with PMSELR.SEL set to n.

If FEAT_FGT is implemented and <n> is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMEVCNTR<n> is as follows:

- If <n> is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and <n> is greater than or equal to the number of accessible event counters, then reads and writes of PMEVCNTR<n> are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if <n> is an UNKNOWN value less-than-or-equal-to the index of the highest accessible event counter.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMEVCNTR<n> are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.P<n> == 0.

Permitted writes of PMEVCNTR<n> are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.{UEN,ER} == {1,1}.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR.{ER,EN} or PMUSERENR_EL0.{UEN,ER,EN}.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, HDCR.HPMN identifies the number of accessible event counters.
- If EL2 is using AArch64, MDCR_EL2.HPMN identifies the number of accessible event counters.

Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see HDCR.HPMN and MDCR_EL2.HPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-30
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b10:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
((IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1[m] == '0' then
            R[t] = Zeros(32);
        else
            R[t] = PMEVCNTR[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
```

```
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMEVCNTR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMEVCNTR[m];
elsif PSTATE.EL == EL3 then
    R[t] = PMEVCNTR[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-30
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b10:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1[m] == '0' || PMUSERENR_EL0.ER ==
'1') then
            return;
        else
            PMEVCNTR[m] = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
```

```
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVCNTR[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVCNTR[m] = R[t];
elsif PSTATE.EL == EL3 then
    PMEVCNTR[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVTYPER<n>, Performance Monitors Event Type Registers, n = 0 - 30

The PMEVTYPER<n> characteristics are:

Purpose

Configures event counter n, where n is 0 to 30.

Configuration

AArch32 System register PMEVTYPER<n> bits [31:0] are architecturally mapped to AArch64 System register PMEVTYPER<n>_EL0[31:0].

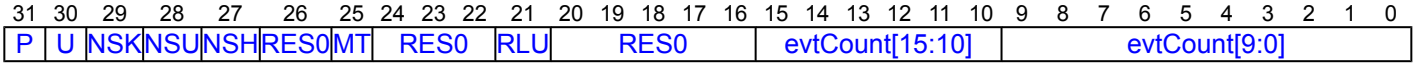
AArch32 System register PMEVTYPER<n> bits [31:0] are architecturally mapped to External register PMEVTYPER<n>_EL0[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMEVTYPER<n> are UNDEFINED.

Attributes

PMEVTYPER<n> is a 32-bit register.

Field descriptions



P, bit [31]

Privileged filtering. Controls counting events in EL1 and, if EL3 is using AArch32, EL3.

P	Meaning
0b0	This mechanism has no effect on filtering of events.
0b1	The PE does not count events in EL1 and, if EL3 is using AArch32, EL3.

If Secure and Non-secure states are implemented, then counting events in Non-secure EL1 is further controlled by PMEVTYPER<n>.NSK.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

User filtering. Controls counting events in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of events.
0b1	The PE does not count events in EL0.

If Secure and Non-secure states are implemented, then counting events in Non-secure EL0 is further controlled by PMEVTYPER<n>.NSU.

If FEAT_RME is implemented, then counting events in Realm EL0 is further controlled by PMEVTYPER<n>.RLU.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting events in Non-secure EL1. If PMEVTYPER<n>.NSK is not equal to PMEVTYPER<n>.P, then the PE does not count events in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of events in Non-secure EL1.

NSK	Meaning
0b0	When PMEVTYPER<n>.P == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>.P == 1, the PE does not count events in Non-secure EL1.
0b1	When PMEVTYPER<n>.P == 0, the PE does not count events in Non-secure EL1. When PMEVTYPER<n>.P == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 filtering. Controls counting events in Non-secure EL0. If PMEVTYPER<n>.NSU is not equal to PMEVTYPER<n>.U, then the PE does not count events in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of events in Non-secure EL0.

NSU	Meaning
0b0	When PMEVTYPER<n>.U == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>.U == 1, the PE does not count events in Non-secure EL0.
0b1	When PMEVTYPER<n>.U == 0, the PE does not count events in Non-secure EL0. When PMEVTYPER<n>.U == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 filtering. Controls counting events in EL2.

NSH	Meaning
0b0	The PE does not count events in EL2.
0b1	This mechanism has no effect on filtering of events.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting events in Secure EL2 is further controlled by PMEVTYPER<n>.SH.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [26]

Reserved, RES0.

MT, bit [25]

When FEAT_MTPMU is implemented or an IMPLEMENTATION DEFINED multi-threaded PMU extension is implemented:

Multithreading.

MT	Meaning
0b0	Count events only on controlling PE.
0b1	Count events from any PE with the same affinity at level 1 and above as this PE.

Unless otherwise stated:

- If the event counts PE cycles when a stall condition is true, then the counter counts Processor cycles when the stall condition is true for all of these PEs.
- If the event counts PE cycles when any other condition is true, then the counter counts Processor cycles when the condition is true for any of these PEs.
- Otherwise, the event counts by the sum of the count across all of these PEs. See 'Multithreaded implementations' and 'Cycle event counting in multithreaded implementations'.

From Armv8.6, the IMPLEMENTATION DEFINED multi-threaded PMU extension is not permitted, meaning if FEAT_MTPMU is not implemented, this field is RES0. See ID_DFR1.MTPMU.

This field is ignored by the PE and treated as zero when FEAT_MTPMU is implemented and disabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [24:22]

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting events in Realm EL0. If PMEVTYPER<n>.RLU is not equal to PMEVTYPER<n>.U, then the PE does not count events in Realm EL0. Otherwise, this mechanism has no effect on filtering of events in Realm EL0.

RLU	Meaning
0b0	When PMEVTYPER<n>.U == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>.U == 1, the PE does not count events in Realm EL0.
0b1	When PMEVTYPER<n>.U == 0, the PE does not count events in Realm EL0. When PMEVTYPER<n>.U == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [20:16]

Reserved, RES0.

evtCount[15:10], bits [15:10]

When FEAT_PMUv3p1 is implemented:

Extension to evtCount[9:0]. For more information, see evtCount[9:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

evtCount[9:0], bits [9:0]

Event to count.

The event number of the event that is counted by event counter PMEVCNTR<n>.

The ranges of event numbers allocated to each type of event are shown in 'Allocation of the PMU event number space'.

If FEAT_PMUv3p8 is implemented and PMEVTYPER<n>.evtCount is programmed to an event that is reserved or not supported by the PE, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>.evtCount field is the value written to the field.

Note

Arm recommends this behavior for all implementations of FEAT_PMUv3.

Otherwise, if PMEVTYPER<n>.evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range `0x0000` to `0x003F`, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>.evtCount field is the value written to the field.
- If FEAT_PMUv3p1 is implemented, for the range `0x4000` to `0x403F`, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>.evtCount field is the value written to the field.
- For other values, it is UNPREDICTABLE what event, if any, is counted and the value returned by a direct or external read of the PMEVTYPER<n>.evtCount field is UNKNOWN.

Note

UNPREDICTABLE means the event must not expose privileged information.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMEVTYPER<n>

PMEVTYPER<n> can also be accessed by using PMXEVTYPER with PMSELR.SEL set to n.

If FEAT_FGT is implemented and <n> is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMEVTYPER<n> is as follows:

- If <n> is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and <n> is greater than or equal to the number of accessible event counters, then reads and writes of PMEVTYPER<n> are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if <n> is an UNKNOWN value less-than-or-equal-to the index of the highest accessible event counter.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMEVTYPER<n> are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.P<n> == 0.

Permitted writes of PMEVTYPER<n> are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.{UEN,ER} == {1,1}.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR.EN or PMUSERENR_EL0.{UEN,EN}.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, HDCR.HPMN identifies the number of accessible event counters.
- If EL2 is using AArch64, MDCR_EL2.HPMN identifies the number of accessible event counters.

Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see HDCR.HPMN and MDCR_EL2.HPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-30
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b11:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1[m] == '0' then
            R[t] = Zeros(32);
        else
            R[t] = PMEVTYPER[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
```

```
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMEVTYPER[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMEVTYPER[m];
elsif PSTATE.EL == EL3 then
    R[t] = PMEVTYPER[m];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>} ; Where m = 0-30
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1110	0b11:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:opc2<2:0>);

if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1[m] == '0' || PMUSERENR_EL0.ER ==
'1') then
            return;
        else
            PMEVTYPER[m] = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
```

```
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVTYPER[m] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVTYPER[m] = R[t];
elsif PSTATE.EL == EL3 then
    PMEVTYPER[m] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENCLR, Performance Monitors Interrupt Enable Clear register

The PMINTENCLR characteristics are:

Purpose

Allows software to disable the generation of interrupt requests on overflows from the following counters:

- The cycle counter PMCCNTR.
- The event counters PMEVCNTR<n>.

Reading from this register shows which overflow interrupt requests are enabled.

Configuration

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to AArch32 System register PMINTENSET[31:0].

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[31:0].

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[31:0].

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to External register PMINTENCLR_EL1[31:0].

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to External register PMINTENSET_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMINTENCLR are UNDEFINED.

Attributes

PMINTENCLR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

C, bit [31]

Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR enable status.

C	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m> disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m> enable status.

P<m>	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m> disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m> enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing PMINTENCLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b010

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMINTENCLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMINTENCLR;
elsif PSTATE.EL == EL3 then
    R[t] = PMINTENCLR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b010

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENCLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENCLR = R[t];
elsif PSTATE.EL == EL3 then
    PMINTENCLR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENSET, Performance Monitors Interrupt Enable Set register

The PMINTENSET characteristics are:

Purpose

Allows software to enable the generation of interrupt requests on overflows from the following counters:

- The cycle counter PMCCNTR.
- The event counters PMEVCNTR<n>.

Reading from this register shows which overflow interrupt requests are enabled.

Configuration

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0].

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[31:0].

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[31:0].

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to External register PMINTENCLR_EL1[31:0].

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to External register PMINTENSET_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMINTENSET are UNDEFINED.

Attributes

PMINTENSET is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

C, bit [31]

Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR enable status.

C	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m> enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m> enable status.

P<m>	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m> disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m> enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing PMINTENSET

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMINTENSET;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMINTENSET;
elsif PSTATE.EL == EL3 then
    R[t] = PMINTENSET;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENSET = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENSET = R[t];
elsif PSTATE.EL == EL3 then
    PMINTENSET = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMMIR, Performance Monitors Machine Identification Register

The PMMIR characteristics are:

Purpose

Describes Performance Monitors parameters specific to the implementation to software.

Configuration

AArch32 System register PMMIR bits [31:0] are architecturally mapped to AArch64 System register PMMIR_EL1[31:0].

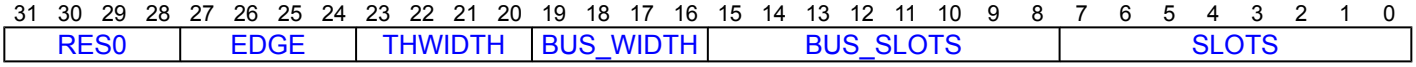
AArch32 System register PMMIR bits [31:0] are architecturally mapped to External register PMMIR[31:0] when FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p4 is implemented.

This register is present only when FEAT_AA32EL1 is implemented and FEAT_PMUv3p4 is implemented. Otherwise, direct accesses to PMMIR are UNDEFINED.

Attributes

PMMIR is a 32-bit register.

Field descriptions



Bits [31:28]

Reserved, RES0.

EDGE, bits [27:24]

PMU event edge detection. With PMMIR.THWIDTH, indicates implementation of event counter thresholding features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EDGE	Meaning
0b0000	FEAT_PMUv3_EDGE is not implemented.
0b0001	FEAT_PMUv3_EDGE is implemented.

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, the only permitted value is 0b0000.

FEAT_PMUv3_EDGE implements the functionality identified by the value 0b0001.

Note

PMEVTYPER<n>_EL0.TE cannot be accessed through PMEVTYPER<n>.

Access to this field is **RO**.

THWIDTH, bits [23:20]

PMEVTYPER<n>_EL0.TH width. Indicates implementation of the FEAT_PMUv3_TH feature, and, if implemented, the size of the PMEVTYPER<n>_EL0.TH field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

THWIDTH	Meaning
0b0000	FEAT_PMUv3_TH is not implemented.
0b0001	1 bit. PMEVTYPER<n>_EL0.TH[11:1] are RES0.
0b0010	2 bits. PMEVTYPER<n>_EL0.TH[11:2] are RES0.
0b0011	3 bits. PMEVTYPER<n>_EL0.TH[11:3] are RES0.
0b0100	4 bits. PMEVTYPER<n>_EL0.TH[11:4] are RES0.
0b0101	5 bits. PMEVTYPER<n>_EL0.TH[11:5] are RES0.
0b0110	6 bits. PMEVTYPER<n>_EL0.TH[11:6] are RES0.
0b0111	7 bits. PMEVTYPER<n>_EL0.TH[11:7] are RES0.
0b1000	8 bits. PMEVTYPER<n>_EL0.TH[11:8] are RES0.
0b1001	9 bits. PMEVTYPER<n>_EL0.TH[11:9] are RES0.
0b1010	10 bits. PMEVTYPER<n>_EL0.TH[11:10] are RES0.
0b1011	11 bits. PMEVTYPER<n>_EL0.TH[11] is RES0.
0b1100	12 bits.

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, this field is zero.

Otherwise, the largest value that can be written to PMEVTYPER<n>_EL0.TH is $2^{(\text{PMMIR.THWIDTH})}$ minus one.

Note

PMEVTYPER<n>_EL0.TH cannot be accessed through PMEVTYPER<n>.

Access to this field is **RO**.

BUS_WIDTH, bits [19:16]

Bus width. Indicates the number of bytes each BUS_ACCESS event relates to. Encoded as Log_2(number of bytes), plus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BUS_WIDTH	Meaning
0b0000	The information is not available.
0b0011	Four bytes.
0b0100	8 bytes.
0b0101	16 bytes.
0b0110	32 bytes.
0b0111	64 bytes.
0b1000	128 bytes.
0b1001	256 bytes.
0b1010	512 bytes.
0b1011	1024 bytes.
0b1100	2048 bytes.

All other values are reserved.

Each transfer is up to this number of bytes. An access might be smaller than the bus width.

When this field is nonzero, each access counted by BUS_ACCESS is at most BUS_WIDTH bytes. An implementation might treat a wide bus as multiple narrower buses, such that a wide access on the bus increments the BUS_ACCESS counter by more than one.

Access to this field is **RO**.

BUS_SLOTS, bits [15:8]

Bus count. The largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle.

When this field is nonzero, the largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle is BUS_SLOTS.

If the bus count information is not available, this field will read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SLOTS, bits [7:0]

Operation width. The largest value by which the STALL_SLOT event might increment by in a single cycle. If the STALL_SLOT event is not implemented, this field might read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMMIR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b110

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_PMUv3p4)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMMIR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMMIR;
elsif PSTATE.EL == EL3 then
    R[t] = PMMIR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVSR, Performance Monitors Overflow Flag Status Register

The PMOVSR characteristics are:

Purpose

Allows software to clear the unsigned overflow flags for the following counters to 0:

- The cycle counter PMCCNTR.
- The event counters PMEVCNTR<n>.

Reading from this register shows the current unsigned overflow flag values.

Configuration

AArch32 System register PMOVSR bits [31:0] are architecturally mapped to AArch32 System register PMOVSSET[31:0].

AArch32 System register PMOVSR bits [31:0] are architecturally mapped to AArch64 System register PMOVSCLR_EL0[31:0].

AArch32 System register PMOVSR bits [31:0] are architecturally mapped to AArch64 System register PMOVSSET_EL0[31:0].

AArch32 System register PMOVSR bits [31:0] are architecturally mapped to External register PMOVSCLR_EL0[31:0].

AArch32 System register PMOVSR bits [31:0] are architecturally mapped to External register PMOVSSET_EL0[31:0].

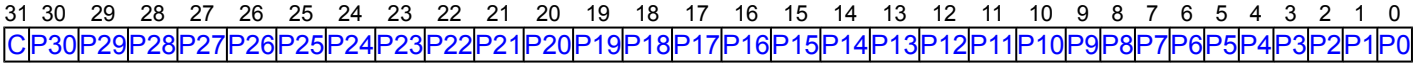
AArch32 System register PMOVSR bits [31:0] are architecturally mapped to External register PMOVS[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMOVSR are UNDEFINED.

Attributes

PMOVSR is a 32-bit register.

Field descriptions



C, bit [31]

Unsigned overflow flag for PMCCNTR clear. On writes, allows software to clear the unsigned overflow flag for PMCCNTR to 0. On reads, returns the unsigned overflow flag for PMCCNTR overflow status.

C	Meaning
0b0	PMCCNTR has not overflowed.
0b1	PMCCNTR has overflowed.

PMCR.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR[31:0] or unsigned overflow of PMCCNTR[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.

        - EL1 is using AArch64.
        - PMUSERENR_EL0.UEN == 1.
        - PMUACR_EL1.C == 0.
    - Access to this field is **RO** if all the following are true:
        - FEAT_PMUv3p9 is implemented.
        - PSTATE.EL == EL0.
        - EL1 is using AArch64.
        - PMUSERENR_EL0.[UEN,CR] == 0b11.
    - Otherwise, access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

Unsigned overflow flag for PMEVCNTR<m> clear. On writes, allows software to clear the unsigned overflow flag for PMEVCNTR<m> to 0. On reads, returns the unsigned overflow flag for PMEVCNTR<m> overflow status.

P<m>	Meaning
0b0	PMEVCNTR<m> has not overflowed.
0b1	PMEVCNTR<m> has overflowed.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP, HDCR.HLP, and PMCR.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>[31:0] or unsigned overflow of PMEVCNTR<m>[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
    - FEAT_PMUv3p9 is implemented.
    - PSTATE.EL == EL0.
    - EL1 is using AArch64.
    - PMUSERENR_EL0.UEN == 1.
    - PMUACR_EL1.P<m> == 0.
- Access to this field is **RO** if all the following are true:
    - FEAT_PMUv3p9 is implemented.
    - PSTATE.EL == EL0.
    - EL1 is using AArch64.
    - PMUSERENR_EL0.[UEN,ER] == 0b11.
- Otherwise, access to this field is **W1C**.

Accessing PMOVSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMOVS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMOVSR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMOVSR;
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMOVSR;
elsif PSTATE.EL == EL3 then
    R[t] = PMOVSR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMOVS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSR = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSR = R[t];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSR = R[t];
elsif PSTATE.EL == EL3 then
    PMOVSR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVSSET, Performance Monitors Overflow Flag Status Set register

The PMOVSSET characteristics are:

Purpose

Allows software to set the unsigned overflow flags for the following counters to 1:

- The cycle counter PMCCNTR.
- The event counters PMEVCNTR<n>.

Reading from this register shows the current unsigned overflow flag values.

Configuration

AArch32 System register PMOVSSET bits [31:0] are architecturally mapped to AArch32 System register PMOVSR[31:0].

AArch32 System register PMOVSSET bits [31:0] are architecturally mapped to AArch64 System register PMOVSSET_EL0[31:0].

AArch32 System register PMOVSSET bits [31:0] are architecturally mapped to AArch64 System register PMOVSCLR_EL0[31:0].

AArch32 System register PMOVSSET bits [31:0] are architecturally mapped to External register PMOVSSET_EL0[31:0].

AArch32 System register PMOVSSET bits [31:0] are architecturally mapped to External register PMOVSCLR_EL0[31:0].

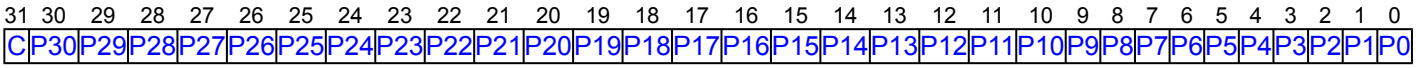
AArch32 System register PMOVSSET bits [31:0] are architecturally mapped to External register PMOVS[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMOVSSET are UNDEFINED.

Attributes

PMOVSSET is a 32-bit register.

Field descriptions



C, bit [31]

Unsigned overflow flag for PMCCNTR set. On writes, allows software to set the unsigned overflow flag for PMCCNTR to 1. On reads, returns the unsigned overflow flag for PMCCNTR overflow status.

C	Meaning
0b0	PMCCNTR has not overflowed.
0b1	PMCCNTR has overflowed.

PMCR.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR[31:0] or unsigned overflow of PMCCNTR[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:

        - FEAT_PMUv3p9 is implemented.
        - PSTATE.EL == EL0.
        - EL1 is using AArch64.
        - PMUSERENR_EL0.UEN == 1.
        - PMUACR_EL1.C == 0.
    - Access to this field is **RO** if all the following are true:
        - FEAT_PMUv3p9 is implemented.
        - PSTATE.EL == EL0.
        - EL1 is using AArch64.
        - PMUSERENR_EL0.[UEN,CR] == 0b11.
    - Otherwise, access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

Unsigned overflow flag for PMEVCNTR<m> set. On writes, allows software to set the unsigned overflow flag for PMEVCNTR<m> to 1. On reads, returns the unsigned overflow flag for PMEVCNTR<m> overflow status.

P<m>	Meaning
0b0	PMEVCNTR<m> has not overflowed.
0b1	PMEVCNTR<m> has overflowed.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP, HDCR.HLP, and PMCR.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>[31:0] or unsigned overflow of PMEVCNTR<m>[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
    - FEAT_PMUv3p9 is implemented.
    - PSTATE.EL == EL0.
    - EL1 is using AArch64.
    - PMUSERENR_EL0.UEN == 1.
    - PMUACR_EL1.P<m> == 0.
- Access to this field is **RO** if all the following are true:
    - FEAT_PMUv3p9 is implemented.
    - PSTATE.EL == EL0.
    - EL1 is using AArch64.
    - PMUSERENR_EL0.[UEN,ER] == 0b11.
- Otherwise, access to this field is **W1S**.

Accessing PMOVSSET

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b011

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMOVS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMOVSSET;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMOVSSET;
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMOVSSET;
elsif PSTATE.EL == EL3 then
    R[t] = PMOVSSET;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b011

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMOVS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSSET = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSSET = R[t];
```

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSSET = R[t];
elsif PSTATE.EL == EL3 then
    PMOVSSET = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSELR, Performance Monitors Event Counter Selection Register

The PMSELR characteristics are:

Purpose

Selects the current event counter PMEVCNTR<n> or the cycle counter PMCCNTR.

Used in conjunction with PMXEVTYPER to determine the event that increments a selected counter, and the modes and states in which the selected counter increments.

Used in conjunction with PMXEVCNTR to determine the value of a selected counter.

Configuration

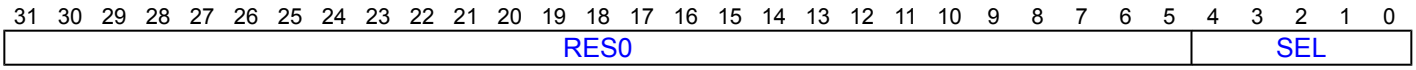
AArch32 System register PMSELR bits [31:0] are architecturally mapped to AArch64 System register PMSELR_EL0[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMSELR are UNDEFINED.

Attributes

PMSELR is a 32-bit register.

Field descriptions



Bits [31:5]

Reserved, RES0.

SEL, bits [4:0]

Event counter select. Selects the counter accessed by subsequent accesses to PMXEVTYPER and PMXEVCNTR.

SEL	Meaning
0b00000..0b11110	Select event counter PMEVCNTR<n>, where n is the value of this field: • MRC and MCR of PMXEVTYPER access PMEVTYPER<n>. • MRC and MCR of PMXEVCNTR access PMEVCNTR<n>.
0b11111	Select the cycle counter, PMCCNTR: • MRC and MCR of PMXEVTYPER access PMCCFILTR. • MRC and MCR of PMXEVCNTR are CONSTRAINED UNPREDICTABLE. For more information, see PMXEVCNTR.

For more information about the results of accesses to the event counters, including when PMSELR.SEL is set to the index of an unimplemented or inaccessible event counter, see PMXEVTYPER and PMXEVCNTR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSELR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
((IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMSELR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMSELR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
```

```
    else
        R[t] = PMSELR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMSELR;
elsif PSTATE.EL == EL3 then
    R[t] = PMSELR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
((IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMSELR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMSELR = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
```

```
    else
        PMSELR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMSELR = R[t];
elsif PSTATE.EL == EL3 then
    PMSELR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSWINC, Performance Monitors Software Increment register

The PMSWINC characteristics are:

Purpose

Increments a counter that is configured to count the Software increment event, event `0x00`. For more information, see 'SW_INCR'.

Configuration

AArch32 System register PMSWINC bits [31:0] are architecturally mapped to AArch64 System register PMSWINC_EL0[31:0].

AArch32 System register PMSWINC bits [31:0] are architecturally mapped to External register PMSWINC_EL0[31:0] when FEAT_PMUv3p9 is not implemented.

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMSWINC are UNDEFINED.

Attributes

PMSWINC is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

Bit [31]

Reserved, RES0.

P<m>, bit [m], for m = 30 to 0

Software increment.

P<m>	Meaning
`0b0`	Write is ignored.
`0b1`	Increment PMEVCNTR<m>, if PMEVCNTR<m> is configured to count software increment events.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3p9 is implemented.
  - PSTATE.EL == EL0.
  - EL1 is using AArch64.
  - PMUSERENR_EL0.[UEN,SW] == 0b10.
  - PMUACR_EL1.P<m> == 0.
- Otherwise, access to this field is **WO/RAZ**.

Accessing PMSWINC

Accesses to this register use the following encodings in the System register encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
((IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,SW,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<SW,EN> == '00')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.<SW,EN> == '00'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMSWINC_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMSWINC = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
```

```
    else
        PMSWINC = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMSWINC = R[t];
elsif PSTATE.EL == EL3 then
    PMSWINC = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMUSERENR, Performance Monitors User Enable Register

The PMUSERENR characteristics are:

Purpose

Enables or disables EL0 access to the Performance Monitors.

Configuration

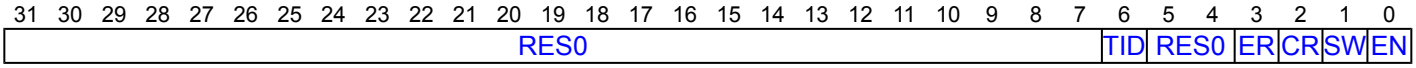
AArch32 System register PMUSERENR bits [31:0] are architecturally mapped to AArch64 System register PMUSERENR_EL0[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMUSERENR are UNDEFINED.

Attributes

PMUSERENR is a 32-bit register.

Field descriptions



Bits [31:7]

Reserved, RES0.

TID, bit [6]

When FEAT_PMUv3p9 is implemented:

Trap ID registers. Traps EL0 read access to common event identification registers.

TID	Meaning
0b0	Accesses to PMCEID<n> are not trapped by this mechanism.
0b1	EL0 read accesses to PMCEID<n> are trapped.

The register accesses affected by this control are:

- MRC reads of PMCEID0, PMCEID1, PMCEID2, and PMCEID3.

When trapped, reads are UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [5:4]

Reserved, RES0.

ER, bit [3]

Event counters Read enable.

When PMUSERENR.EN is 0, PMUSERENR.ER enables EL0 reads of the event counters and EL0 reads and writes of the select register.

ER	Meaning
0b0	EL0 reads of the event counters and EL0 reads and writes of the select register are disabled, unless enabled by PMUSERENR.EN.
0b1	EL0 reads of the event counters and EL0 reads and writes of the select register are enabled, unless trapped by another control.

The register accesses affected by this control are:

- MRC reads of PMEVCNTR<n> and PMXEVCNTR.
- MRC and MCR accesses to PMSELR.

When disabled, reads and writes are UNDEFINED.

This field is ignored by the PE when PMUSERENR.EN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CR, bit [2]

Cycle counter Read enable.

When PMUSERENR.EN is 0, PMUSERENR.CR enables EL0 reads of the cycle counter.

CR	Meaning
0b0	EL0 reads of the cycle counter are disabled, unless enabled by PMUSERENR.EN.
0b1	EL0 reads of the cycle counter are enabled, unless trapped by another control.

The register accesses affected by this control are:

- MRC reads of PMCCNTR.
- MRRC reads of PMCCNTR.

When disabled, reads are UNDEFINED.

This field is ignored by the PE when PMUSERENR.EN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SW, bit [1]

Software increment register Write enable.

When PMUSERENR.EN is 0, PMUSERENR.SW enables EL0 writes to the Software increment register.

SW	Meaning
0b0	EL0 writes to the Software increment register are disabled, unless enabled by PMUSERENR.EN.
0b1	EL0 writes to the Software increment register are enabled, unless trapped by another control.

The register accesses affected by this control are:

- MCR writes to PMSWINC.

When disabled, writes are UNDEFINED.

This field is ignored by the PE when PMUSERENR.EN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EN, bit [0]

Enable. Enables EL0 read/write access to PMU registers.

EN	Meaning
0b0	EL0 accesses to the specified PMU System registers are trapped, unless enabled by PMUSERENR.{ER,CR,SW}.
0b1	EL0 accesses to the specified PMU System registers are enabled, unless trapped by another control.

The register accesses affected by this control are:

- MRC or MCR accesses to PMCCFILTR, PMCCNTR, PMCNTENCLR, PMCNTENSET, PMCR, PMEVCNTR<n>, PMEVTYPER<n>, PMOVSR, PMOVSSET, PMSELR, PMXEVCNTR, and PMXEVTYPER.
- MRC reads of the following registers:
  - PMCEID0 and PMCEID1.
  - If FEAT_PMUv3p1 is implemented, PMCEID2 and PMCEID3.
- MCR writes to PMSWINC.
- MRRC or MCRR accesses to PMCCNTR.

When trapped, reads and writes are UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMUSERENR

When FEAT_PMUv3p9 is implemented and EL1 is using AArch64, PMUSERENR_EL0 contains additional controls that affect the behavior of this register.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b000

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMUSERENR;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMUSERENR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMUSERENR;
elsif PSTATE.EL == EL3 then
    R[t] = PMUSERENR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1110	0b000

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMUSERENR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMUSERENR = R[t];
elsif PSTATE.EL == EL3 then
    PMUSERENR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMXEVCNTR, Performance Monitors Selected Event Count Register

The PMXEVCNTR characteristics are:

Purpose

Reads or writes the value of the selected event counter, PMEVCNTR<n>. PMSELR.SEL determines which event counter is selected.

Configuration

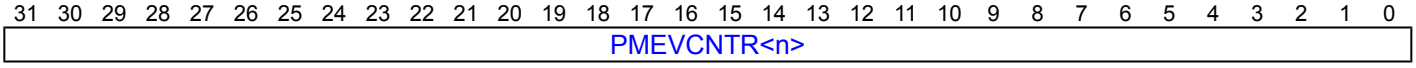
AArch32 System register PMXEVCNTR bits [31:0] are architecturally mapped to AArch64 System register PMXEVCNTR_EL0[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMXEVCNTR are UNDEFINED.

Attributes

PMXEVCNTR is a 32-bit register.

Field descriptions



PMEVCNTR<n>, bits [31:0]

Value of the selected event counter, PMEVCNTR<n>, where n is the value stored in PMSELR.SEL.

If FEAT_PMUv3p5 is implemented, the event counter is 64 bits and only the least-significant part of the event counter is accessible in AArch32 state:

- Reads from PMXEVCNTR return bits [31:0] of the counter.
- Writes to PMXEVCNTR update bits [31:0] and leave bits [63:32] unchanged.
- There is no means to access bits [63:32] directly from AArch32 state.
- If the implementation does not support AArch64, bits [63:32] are not required to be implemented.

If FEAT_PMUv3p5 is not implemented, the event counter is 32 bits.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMXEVCNTR

If FEAT_FGT is implemented and PMSELR.SEL is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMXEVCNTR is as follows:

- If PMSELR.SEL is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and PMSELR.SEL is greater than or equal to the number of accessible event counters, then reads and writes of PMXEVCNTR are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.

- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if PMSELR.SEL has an UNKNOWN value less than the number of event counters accessible at the current Exception level and Security state.
- If EL2 is implemented and enabled in the current Security state, and PMSELR.SEL is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMXEVCNTR are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.P<UInt(PMSELR.SEL)> == 0.

Permitted writes of PMXEVCNTR are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.{UEN,ER} == {1,1}.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR.{ER,EN} or PMUSERENR_EL0.{UEN,ER,EN}.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, HDCR.HPMN identifies the number of accessible event counters.
- If EL2 is using AArch64, MDCR_EL2.HPMN identifies the number of accessible event counters.

Otherwise, the number of accessible event counters is determined by the Effective value of PMCCR.EPMN. For more information, see HDCR.HPMN, MDCR_EL2.HPMN and PMCCR.EPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1101	0b010

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif UInt(PMSELR.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) &&
((IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00'
then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && UInt(PMSELR.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1[UInt(PMSELR.SEL)] == '0' then
            R[t] = Zeros(32);
        else
            R[t] = PMEVCNTR[UInt(PMSELR.SEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
```

```
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && UInt(PMSELR.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMEVCNTR[UInt(PMSELR.SEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = PMEVCNTR[UInt(PMSELR.SEL)];
elsif PSTATE.EL == EL3 then
    R[t] = PMEVCNTR[UInt(PMSELR.SEL)];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1101	0b010

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif UInt(PMSELR.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && UInt(PMSELR.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1[UInt(PMSELR.SEL)] == '0' ||
PMUSERENR_EL0.ER == '1') then
            return;
        else
            PMEVCNTR[UInt(PMSELR.SEL)] = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
```

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && UInt(PMSELR.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVCNTR[UInt(PMSELR.SEL)] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVCNTR[UInt(PMSELR.SEL)] = R[t];
elsif PSTATE.EL == EL3 then
    PMEVCNTR[UInt(PMSELR.SEL)] = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMXEVTYPER, Performance Monitors Selected Event Type Register

The PMXEVTYPER characteristics are:

Purpose

When PMSELR.SEL selects an event counter, this accesses a PMEVTYPER<n> register. When PMSELR.SEL selects the cycle counter, this accesses PMCCFILTR.

Configuration

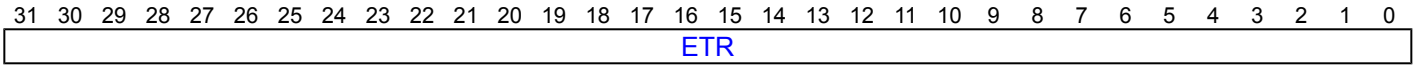
AArch32 System register PMXEVTYPER bits [31:0] are architecturally mapped to AArch64 System register PMXEVTYPER_EL0[31:0].

This register is present only when FEAT_AA32 is implemented and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMXEVTYPER are UNDEFINED.

Attributes

PMXEVTYPER is a 32-bit register.

Field descriptions



ETR, bits [31:0]

Event type register or PMCCFILTR.

When PMSELR.SEL == 31, this register accesses PMCCFILTR.

Otherwise, this register accesses PMEVTYPER<n> where n is the value in PMSELR.SEL.

Accessing PMXEVTYPER

If FEAT_FGT is implemented, and PMSELR.SEL is not 31 and is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMXEVTYPER is as follows:

- If PMSELR.SEL is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented, and PMSELR.SEL is not 31 and is greater than or equal to the number of accessible event counters, then reads and writes of PMXEVTYPER are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if PMSELR.SEL has an UNKNOWN value less than the number of event counters accessible at the current Exception level and Security state.
- Accesses to the register behave as if PMSELR.SEL is 31.
- If EL2 is implemented and enabled in the current Security state, and PMSELR.SEL is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMXEVTYPER are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.

- EL1 is using AArch64.
- PMUSERENR_EL0.UEN == 1.
- Any of the following are true:
    - PMSELR.SEL != 31 and PMUACR_EL1.P<UInt(PMSELR.SEL)> == 0.
    - PMSELR.SEL == 31 and PMUACR_EL1.C == 0.

Permitted writes of PMXEVTYPER are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- EL1 is using AArch64.
- PMUSERENR_EL0.UEN == 1.
- Any of the following are true:
    - PMSELR.SEL != 31 and PMUSERENR_EL0.ER == 1.
    - PMSELR.SEL == 31 and PMUSERENR_EL0.CR == 1.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR.EN or PMUSERENR_EL0.{UEN,EN}.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, HDCR.HPMN identifies the number of accessible event counters.
- If EL2 is using AArch64, MDCR_EL2.HPMN identifies the number of accessible event counters.

Otherwise, the number of accessible event counters is determined by the Effective value of PMCCR.EPMN. For more information, see HDCR.HPMN, MDCR_EL2.HPMN and PMCCR.EPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1101	0b001

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif UInt(PMSELR.SEL) != 31 && UInt(PMSELR.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && UInt(PMSELR.SEL) != 31 && UInt(PMSELR.SEL) >=
GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && ((UInt(PMSELR.SEL) != 31 &&
PMUACR_EL1[UInt(PMSELR.SEL)] == '0') || (UInt(PMSELR.SEL) == 31 && PMUACR_EL1.C == '0')) then
            R[t] = Zeros(32);
        elsif UInt(PMSELR.SEL) == 31 then
            R[t] = PMCCFILTR;
        else
            R[t] = PMEVTYPER[UInt(AArch32-PMSELR.SEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
```

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && UInt(PMSELR.SEL) != 31 && UInt(PMSELR.SEL) >=
GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if UInt(PMSELR.SEL) == 31 then
            R[t] = PMCCFILTR;
        else
            R[t] = PMEVTYPER[UInt(AArch32-PMSELR.SEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if UInt(PMSELR.SEL) == 31 then
            R[t] = PMCCFILTR;
        else
            R[t] = PMEVTYPER[UInt(AArch32-PMSELR.SEL)];
elsif PSTATE.EL == EL3 then
    if UInt(PMSELR.SEL) == 31 then
        R[t] = PMCCFILTR;
    else
        R[t] = PMEVTYPER[UInt(AArch32-PMSELR.SEL)];
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1001	0b1101	0b001

```
if !(IsFeatureImplemented(FEAT_AA32) && IsFeatureImplemented(FEAT_PMUv3)) then
    UNDEFINED;
elsif UInt(PMSELR.SEL) != 31 && UInt(PMSELR.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1) && (PMUSERENR_EL0.EN == '0'
&& (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')) then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif IsFeatureImplemented(FEAT_AA32EL1) && ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) &&
HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T9
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HDCR.TPM
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && UInt(PMSELR.SEL) != 31 && UInt(PMSELR.SEL) >=
GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64EL1) &&
!ELUsingAArch32(EL2) && PMUSERENR_EL0.UEN == '1' && ((UInt(PMSELR.SEL) != 31 &&
(PMUACR_EL1[UInt(PMSELR.SEL)] == '0' || PMUSERENR_EL0.ER == '1')) || (UInt(PMSELR.SEL) == 31 &&
(PMUACR_EL1.C == '0' || PMUSERENR_EL0.CR == '1'))) then
            return;
        elsif UInt(PMSELR.SEL) == 31 then
            PMCCFILTR = R[t];
        else
            PMEVTYPER[UInt(AArch32-PMSELR.SEL)] = R[t];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
```

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T9 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TPM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && UInt(PMSELR.SEL) != 31 && UInt(PMSELR.SEL) >=
GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        elsif IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
            AArch32.TakeHypTrapException(0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if UInt(PMSELR.SEL) == 31 then
            PMCCFILTR = R[t];
        else
            PMEVTYPER[UInt(AArch32-PMSELR.SEL)] = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        if UInt(PMSELR.SEL) == 31 then
            PMCCFILTR = R[t];
        else
            PMEVTYPER[UInt(AArch32-PMSELR.SEL)] = R[t];
elsif PSTATE.EL == EL3 then
    if UInt(PMSELR.SEL) == 31 then
        PMCCFILTR = R[t];
    else
        PMEVTYPER[UInt(AArch32-PMSELR.SEL)] = R[t];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PRRR, Primary Region Remap Register

The PRRR characteristics are:

Purpose

Controls the top-level mapping of the TEX[0], C, and B memory region attributes.

Configuration

This register is banked between PRRR and PRRR_S and PRRR_NS.

AArch32 System register PRRR bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL1[31:0] when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register PRRR bits [31:0] are architecturally mapped to AArch32 System register MAIR0[31:0] when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register PRRR bits [31:0] (PRRR_S) are architecturally mapped to AArch32 System register MAIR0[31:0] (MAIR0_S) when EL3 is using AArch32.

AArch32 System register PRRR bits [31:0] (PRRR_NS) are architecturally mapped to AArch32 System register MAIR0[31:0] (MAIR0_NS) when EL3 is using AArch32.

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to PRRR are UNDEFINED.

MAIR0 and PRRR are the same register, with a different view depending on the value of TTBCR.EAE:

- When it is set to 0, the register is as described in PRRR.
- When it is set to 1, the register is as described in MAIR0.

Attributes

PRRR is a 32-bit register.

This register has the following instances:

- PRRR, when EL3 is not implemented or FEAT_AA64 is implemented.
- PRRR_S, when FEAT_AA32EL3 is implemented.
- PRRR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When TTBCR.EAE == 0:

31	30	29	28	27	26	25	24	23 22 21 20	19	18	17	16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
NOS7	NOS6	NOS5	NOS4	NOS3	NOS2	NOS1	NOS0	RES0	NS1	NS0	DS1	DS0	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0

NOS<n>, bit [n+24], for n = 7 to 0

Not Outer Shareable. NOS<n> is the Outer Shareable property for memory attributes n, if the region is mapped as Normal memory that is not Inner Non-cacheable, Outer Non-cacheable, and the appropriate PRRR.{NS0, NS1} field identifies the region as shareable. n is the value of the concatenation of the {TEX[0], C, B} bits from the Translation table descriptor. The possible values of each NOS<n> field other than NOS6 are:

NOS<n>	Meaning
0b0	Memory region is Outer Shareable.
0b1	Memory region is Inner Shareable.

The value of this bit is ignored if the region is:

- Device memory
- Normal memory that is at least one of:
    - Inner Non-cacheable, Outer Non-cacheable.
    - Identified by the appropriate PRRR.{NS0, NS1} field as Non-shareable.

The meaning of the NOS6 field is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:20]

Reserved, RES0.

NS1, bit [19]

Mapping of S = 1 attribute for Normal memory regions. This field is used in determining the Shareability of a memory region that is mapped to Normal memory and both:

- Is not Inner Non-cacheable, Outer Non-cacheable.
- Has the S bit in the Translation table descriptor set to 1.

NS1	Meaning
0b0	Region is Non-shareable.
0b1	Region is shareable. The value of the appropriate PRRR.NOS<n> field determines whether the region is Inner Shareable or Outer Shareable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS0, bit [18]

Mapping of S = 0 attribute for Normal memory regions. This field is used in determining the Shareability of a memory region that is mapped to Normal memory and both:

- Is not Inner Non-cacheable, Outer Non-cacheable.
- Has the S bit in the Translation table descriptor set to 0.

NS0	Meaning
0b0	Region is Non-shareable.
0b1	Region is shareable. The value of the appropriate PRRR.NOS<n> field determines whether the region is Inner Shareable or Outer Shareable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DS1, bit [17]

Mapping of S = 1 attribute for Device memory. From Armv8.0, all types of Device memory are Outer Shareable, and therefore this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DS0, bit [16]

Mapping of S = 0 attribute for Device memory. From Armv8.0, all types of Device memory are Outer Shareable, and therefore this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TR<n>, bits [2n+1:2n], for n = 7 to 0

TR<n> is the primary TEX mapping for memory attributes n, and defines the mapped memory type for a region with attributes n. n is the value of the concatenation of the {TEX[0], C, B} bits from the Translation table descriptor. The possible values for each field other than TR6 are:

TR<n>	Meaning
0b00	Device-nGnRnE memory
0b01	Device-nGnRE memory
0b10	Normal memory

The value 0b11 is reserved. The effect of programming a field to 0b11 is CONSTRAINED UNPREDICTABLE.

The meaning of the TR6 field is IMPLEMENTATION DEFINED.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PRRR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            R[t] = MAIR0_NS;
        else
            R[t] = PRRR_NS;
    else
        if TTBCR.EAE == '1' then
            R[t] = MAIR0;
        else
            R[t] = PRRR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            R[t] = MAIR0_NS;
        else
            R[t] = PRRR_NS;
    else
        if TTBCR.EAE == '1' then
            R[t] = MAIR0;
        else
            R[t] = PRRR;
elsif PSTATE.EL == EL3 then
    if TTBCR.EAE == '1' then
        if SCR.NS == '0' then
            R[t] = MAIR0_S;
        else
            R[t] = MAIR0_NS;
    else
        if SCR.NS == '0' then
            R[t] = PRRR_S;
        else
            R[t] = PRRR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T10
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T10 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR0_NS = R[t];
        else
            PRRR_NS = R[t];
    else
        if TTBCR.EAE == '1' then
            MAIR0 = R[t];
        else
            PRRR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR0_NS = R[t];
        else
            PRRR_NS = R[t];
    else
        if TTBCR.EAE == '1' then
            MAIR0 = R[t];
        else
            PRRR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if TTBCR.EAE == '1' then
            if SCR.NS == '0' then
                MAIR0_S = R[t];
            else
                MAIR0_NS = R[t];
        else
            if SCR.NS == '0' then
                PRRR_S = R[t];
            else
                PRRR_NS = R[t];
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

REVIDR, Revision ID Register

The REVIDR characteristics are:

Purpose

Provides implementation-specific minor revision information.

Configuration

AArch32 System register REVIDR bits [31:0] are architecturally mapped to AArch64 System register REVIDR_EL1[31:0].

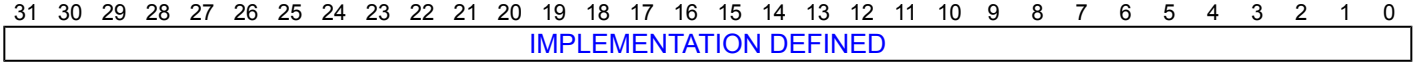
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to REVIDR are UNDEFINED.

If REVIDR has the same value as MIDR, then its contents have no significance.

Attributes

REVIDR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing REVIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0000	0b110

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = REVIDR;
elsif PSTATE.EL == EL2 then
    R[t] = REVIDR;
elsif PSTATE.EL == EL3 then
    R[t] = REVIDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RMR, Reset Management Register

The RMR characteristics are:

Purpose

If EL1 or EL3 is the highest implemented Exception level and this register is implemented:

- A write to the register at the highest implemented Exception level can request a Warm reset.
- If the highest implemented Exception level can use AArch32 and AArch64, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch32 System register RMR bits [31:0] are architecturally mapped to AArch64 System register RMR_EL1[31:0] when the highest implemented Exception level is EL1.

AArch32 System register RMR bits [31:0] are architecturally mapped to AArch64 System register RMR_EL3[31:0] when EL3 is implemented.

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to RMR are UNDEFINED.

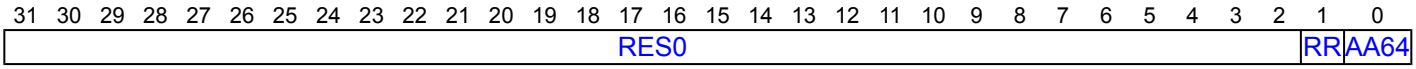
Only implemented if EL1 or EL3 is the highest implemented Exception level. In this case:

- If the highest implemented Exception level can use AArch32 and AArch64 then this register must be implemented.
- If the highest implemented Exception level cannot use AArch64 then it is IMPLEMENTATION DEFINED whether the register is implemented.

Attributes

RMR is a 32-bit register.

Field descriptions



Bits [31:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

AA64, bit [0]

When the highest implemented Exception level can use AArch64, determines which Execution state the PE boots into after a Warm reset:

AA64	Meaning
0b0	AArch32.
0b1	AArch64.

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If the highest implemented Exception level cannot use AArch64 this bit is RAZ/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing RMR

When EL3 is implemented, Arm deprecates accessing this register from any PE mode other than Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL IN {EL1, EL3} && IsHighestEL(PSTATE.EL) then
    R[t] = RMR;
else
    UNDEFINED;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsHighestEL(EL1) then
        RMR = R[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        RMR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RVBAR, Reset Vector Base Address Register

The RVBAR characteristics are:

Purpose

If EL3 is not implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch32 state.

Configuration

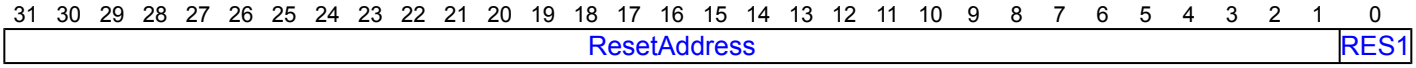
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to RVBAR are UNDEFINED.

This register is implemented only if the highest Exception level implemented is capable of using AArch32, and is not EL3.

Attributes

RVBAR is a 32-bit register.

Field descriptions



ResetAddress, bits [31:1]

Bits [31:1] of the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 32-bit state.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [0]

Reserved, RES1.

Accessing RVBAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsHighestEL(EL1) then
        R[t] = RVBAR;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsHighestEL(EL2) then
        R[t] = RVBAR;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    R[t] = MVBAR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCR, Secure Configuration Register

The SCR characteristics are:

Purpose

When EL3 is implemented and can use AArch32, defines the configuration of the current Security state. It specifies:

- The Security state, either Secure or Non-secure.
- What mode the PE branches to if an IRQ, FIQ, or External abort occurs.
- Whether the PSTATE.F or PSTATE.A bits can be modified when SCR.NS==1.

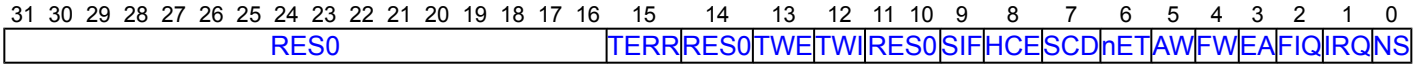
Configuration

This register is present only when FEAT_AA32EL3 is implemented. Otherwise, direct accesses to SCR are UNDEFINED.

Attributes

SCR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

TERR, bit [15]

When FEAT_RAS is implemented:

Trap Error record accesses. Generate a Monitor Trap exception on MRC, MCR, MRRC, or MCRR accesses to the following registers from modes other than Monitor mode, reported using EC syndrome values `0x03` and `0x04`:

ERRIDR, ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXFR, ERXFR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS. When FEAT_RASv1p1 is implemented, ERXMISC4, ERXMISC5, ERXMISC6, ERXMISC7.

TERR	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Accesses to the specified registers from modes other than Monitor mode generate a Monitor Trap exception.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

Bit [14]

Reserved, RES0.

TWE, bit [13]

Traps WFE instructions to Monitor mode.

TWE	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFE instruction in any mode other than Monitor mode is trapped to Monitor mode, if the instruction would otherwise have caused the PE to enter a low-power state and the attempted execution does not generate an exception that is taken to EL1 or EL2 by SCTLR.nTWE or HCR.TWE. Any exception that is taken to EL1 or to EL2 has priority over this trap.

The attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

TWI, bit [12]

Traps WFI instructions to Monitor mode.

TWI	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Any attempt to execute a WFI instruction in any mode other than Monitor mode is trapped to Monitor mode, if the instruction would otherwise have caused the PE to enter a low-power state and the attempted execution does not generate an exception that is taken to EL1 or EL2 by SCTLR.nTWI or HCR.TWI. Any exception that is taken to EL1 or to EL2 has priority over this trap.

The attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Bits [11:10]

Reserved, RES0.

SIF, bit [9]

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction execution from Non-secure memory.

SIF	Meaning
0b0	Secure state instruction execution from Non-secure memory is permitted.
0b1	Secure state instruction execution from Non-secure memory is not permitted.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

HCE, bit [8]

Hypervisor Call instruction enable. If EL2 is implemented, enables execution of HVC instructions at Non-secure EL1 and EL2.

HCE	Meaning
0b0	HVC instructions are: • UNDEFINED at Non-secure EL1. The Undefined Instruction exception is taken from PL1 to PL1. • UNPREDICTABLE at EL2. Behavior is one of the following: ◦ The instruction is UNDEFINED. ◦ The instruction executes as a NOP.
0b1	HVC instructions are enabled at Non-secure EL1 and EL2.

Note

HVC instructions are always UNDEFINED at EL0 and in Secure state.

If EL2 is not implemented, this bit is RES0 and HVC is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

SCD, bit [7]

Secure Monitor Call disable. Disables SMC instructions.

SCD	Meaning
0b0	SMC instructions are enabled.
0b1	In Non-secure state, SMC instructions are UNDEFINED. The Undefined Instruction exception is taken from the current Exception level to the current Exception level. In Secure state, behavior is one of the following: • The instruction is UNDEFINED. • The instruction executes as a NOP.

Note

SMC instructions are always UNDEFINED at PL0.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

nET, bit [6]

Not Early Termination. This bit disables early termination.

nET	Meaning
0b0	Early termination permitted. Execution time of data operations can depend on the data values.
0b1	Disable early termination. The number of cycles required for data operations is forced to be independent of the data values.

This IMPLEMENTATION DEFINED mechanism can disable data dependent timing optimizations from multiplies and data operations. It can provide system support against information leakage that might be exploited by timing correlation types of attack.

On implementations that do not support early termination or do not support disabling early termination, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

AW, bit [5]

When the value of SCR.EA is 1 and the value of HCR.AMO is 0, this bit controls whether PSTATE.A masks an External abort taken from Non-secure state.

AW	Meaning
0b0	External aborts taken from Non-secure state are not masked by PSTATE.A, and are taken to EL3. External aborts taken from Secure state are masked by PSTATE.A.
0b1	External aborts taken from either Security state are masked by PSTATE.A. When PSTATE.A is 0, the abort is taken to EL3.

When SCR.EA is 0 or HCR.AMO is 1, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

FW, bit [4]

When the value of SCR.FIQ is 1 and the value of HCR.FMO is 0, this bit controls whether PSTATE.F masks an FIQ interrupt taken from Non-secure state.

FW	Meaning
0b0	An FIQ taken from Non-secure state is not masked by PSTATE.F, and is taken to EL3. An FIQ taken from Secure state is masked by PSTATE.F.
0b1	An FIQ taken from either Security state is masked by PSTATE.F. When PSTATE.F is 0, the FIQ is taken to EL3.

When SCR.FIQ is 0 or HCR.FMO is 1, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

EA, bit [3]

External Abort handler. This bit controls which mode takes External aborts and SError exceptions.

EA	Meaning
0b0	External aborts taken to Abort mode.
0b1	External aborts taken to Monitor mode.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

FIQ, bit [2]

FIQ handler. This bit controls which mode takes FIQ exceptions.

FIQ	Meaning
0b0	FIQs taken to FIQ mode.
0b1	FIQs taken to Monitor mode.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

IRQ, bit [1]

IRQ handler. This bit controls which mode takes IRQ exceptions.

IRQ	Meaning
0b0	IRQs taken to IRQ mode.
0b1	IRQs taken to Monitor mode.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

NS, bit [0]

Non-secure bit. Except when the PE is in Monitor mode, this bit determines the Security state of the PE:

NS	Meaning
0b0	PE is in Secure state.
0b1	PE is in Non-secure state.

If the HCR.TGE bit is set, an attempt to change from a Secure PL1 mode to a Non-secure EL1 mode by changing the SCR.NS bit from 0 to 1 results in the SCR.NS bit remaining as 0.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Accessing SCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL3) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    R[t] = SCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL3) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR, System Control Register

The SCTLR characteristics are:

Purpose

Provides the top-level control of the system, including its memory system.

Configuration

This register is banked between SCTLR and SCTLR_S and SCTLR_NS.

AArch32 System register SCTLR bits [31:0] are architecturally mapped to AArch64 System register SCTLR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to SCTLR are UNDEFINED.

Some bits in the register are read-only. These bits relate to non-configurable features of an implementation, and are provided for compatibility with previous versions of the architecture.

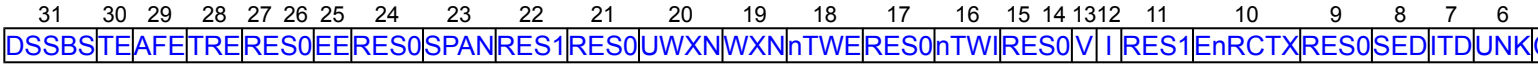
Attributes

SCTLR is a 32-bit register.

This register has the following instances:

- SCTLR, when EL3 is not implemented or FEAT_AA64 is implemented.
- SCTLR_S, when FEAT_AA32EL3 is implemented.
- SCTLR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



DSSBS, bit [31]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry. The defined values are:

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to any mode in this security state except Hyp mode
0b1	PSTATE.SSBS is set to 1 on an exception to any mode in this security state except Hyp mode

Note

When EL3 is implemented and is using AArch32, this bit is banked between the two Security states.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

TE, bit [30]

T32 Exception Enable. This bit controls whether exceptions to an Exception level that is executing at PL1 are taken to A32 or T32 state:

TE	Meaning
0b0	Exceptions, including reset, taken to A32 state.
0b1	Exceptions, including reset, taken to T32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

AFE, bit [29]

Access Flag Enable. When using the Short-descriptor translation table format for the PL1&0 translation regime, this bit enables use of the AP[0] bit in the translation descriptors as the Access flag, and restricts access permissions in the translation descriptors to the simplified model.

AFE	Meaning
0b0	In the Translation table descriptors, AP[0] is an access permissions bit. The full range of access permissions is supported. No Access flag is implemented.
0b1	In the Translation table descriptors, AP[0] is the Access flag. Only the simplified model for access permissions is supported.

When using the Long-descriptor translation table format, the VMSA behaves as if this bit is set to 1, regardless of the value of this bit.

The AFE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TRE, bit [28]

TEX remap enable. This bit enables remapping of the TEX[2:1] bits in the PL1&0 translation regime for use as two translation table bits that can be managed by the operating system. Enabling this remapping also changes the scheme used to describe the memory region attributes in the VMSA.

TRE	Meaning
0b0	TEX remap disabled. TEX[2:0] are used, with the C and B bits, to describe the memory region attributes.
0b1	TEX remap enabled. TEX[2:1] are reassigned for use as bits managed by the operating system. The TEX[0], C, and B bits are used to describe the memory region attributes, with the MMU remap registers.

When the value of TTBCR.EAE is 1, this bit is RES1.

The TRE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [27:26]

Reserved, RES0.

EE, bit [25]

The value of the PSTATE.E bit on branch to an exception vector or coming out of reset, and the endianness of stage 1 translation table walks in the PL1&0 translation regime.

EE	Meaning
0b0	Little-endian. PSTATE.E is cleared to 0 on taking an exception or coming out of reset. Stage 1 translation table walks in the PL1&0 translation regime are little-endian.
0b1	Big-endian. PSTATE.E is set to 1 on taking an exception or coming out of reset. Stage 1 translation table walks in the PL1&0 translation regime are big-endian.

If an implementation does not provide Big-endian support for data accesses at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support for data accesses at Exception levels higher than EL0, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bit [24]

Reserved, RES0.

SPAN, bit [23]

When FEAT_PAN is implemented:

Set Privileged Access Never, on taking an exception to EL1 from either Secure or Non-secure state, or to EL3 from Secure state when EL3 is using AArch32.

SPAN	Meaning
0b0	PSTATE.PAN is set to 1 in the following situations: • In Non-secure state, on taking an exception to EL1. • In Secure state, when EL3 is using AArch64, on taking an exception to EL1. • In Secure state, when EL3 is using AArch32, on taking an exception to EL3.
0b1	The value of PSTATE.PAN is left unchanged on taking an exception to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [22]

Reserved, RES1.

Bit [21]

Reserved, RES0.

UWXN, bit [20]

Unprivileged write permission implies PL1 XN (Execute-never). This bit can force all memory regions that are writable at PL0 to be treated as XN for accesses from software executing at PL1.

UWXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable at PL0 forced to XN for accesses from software executing at PL1.

The UWXN bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

WXN, bit [19]

Write permission implies XN (Execute-never). For the PL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the PL1&0 translation regime is forced to XN for accesses from software executing at PL1 or PL0.

This bit applies only when SCTLR.M bit is set.

The WXN bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

nTWE, bit [18]

Traps EL0 execution of WFE instructions to Undefined mode.

nTWE	Meaning
0b0	Any attempt to execute a WFE instruction at EL0 is trapped to Undefined mode, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

The attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Traps EL0 execution of WFI instructions to Undefined mode.

nTWI	Meaning
0b0	Any attempt to execute a WFI instruction at EL0 is trapped to Undefined mode, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

The attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no

Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'1'`.

Bits [15:14]

Reserved, RES0.

V, bit [13]

Vectors bit. This bit selects the base address of the exception vectors for exceptions taken to a PE mode other than Monitor mode or Hyp mode:

V	Meaning
`0b0`	Normal exception vectors. Base address is held in VBAR.
`0b1`	High exception vectors (Hivecs), base address `0xFFFF0000`. This base address cannot be remapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

I, bit [12]

Instruction access Cacheability control, for accesses at EL1 and EL0:

I	Meaning
`0b0`	All instruction access to Normal memory from PL1 and PL0 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR.M is 0, instruction accesses from stage 1 of the PL1&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
`0b1`	All instruction access to Normal memory from PL1 and PL0 can be cached at all levels of instruction and unified cache. If the value of SCTLR.M is 0, instruction accesses from stage 1 of the PL1&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

Instruction accesses to Normal memory from EL1 and EL0 are Cacheable regardless of the value of the SCTLR.I bit if either:

- EL2 is using AArch32 and the value of HCR.DC is 1.
- EL2 is using AArch64 and the value of HCR_EL2.DC is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Bit [11]

Reserved, RES1.

EnRCTX, bit [10]

When FEAT_SPECRES is implemented:

Enable EL0 access to the following System instructions:

- CFPRCTX.
- DVPRCTX.
- CPPRCTX.
- If FEAT_SPECRES2 is implemented, COSPRCTX.

EnRCTX	Meaning
0b0	EL0 access to these instructions is disabled, and these instructions are trapped to EL1.
0b1	EL0 access to these instructions is enabled.

Note

When EL3 is implemented and is using AArch32, this bit is banked between the two Security states.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [9]

Reserved, RES0.

SED, bit [8]

SETEND instruction disable. Disables SETEND instructions at PL0 and PL1.

SED	Meaning
0b0	SETEND instruction execution is enabled at PL0 and PL1.
0b1	SETEND instructions are UNDEFINED at PL0 and PL1.

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

ITD, bit [7]

IT Disable. Disables some uses of IT instructions at PL1 and PL0.

ITD	Meaning
0b0	All IT instruction functionality is enabled at PL1 and PL0.
0b1	Any attempt at PL1 or PL0 to execute any of the following is UNDEFINED: • All encodings of the IT instruction with hw1[3:0]!=1000. • All encodings of the subsequent instruction with the following values for hw1: ◦ 0b11xxxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. ◦ 0b1011xxxxxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions'. ◦ 0b10100xxxxxxxxxx: ADD Rd, PC, #imm ◦ 0b01001xxxxxxxxxx: LDR Rd, [PC, #imm] ◦ 0b0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. ◦ 0b010001xx1xxxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn. These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block. It is IMPLEMENTATION DEFINED whether the IT instruction is treated as: • A 16-bit instruction, that can only be followed by another 16-bit instruction. • The first half of a 32-bit instruction. This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED. An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information see 'Changes to an ITD control by an instruction in an IT block'.

ITD is optional, but if it is implemented in the SCTLR then it must also be implemented in the SCTLR_EL1, SCTLR_EL2, and HSCTLR.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When an implementation does not implement ITD, access to this field is **RAZ/WI**.

UNK, bit [6]

Writes to this bit are IGNORED. Reads of this bit return an UNKNOWN value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CP15BEN, bit [5]

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from PL1 and PL0:

CP15BEN	Meaning
0b0	PL0 and PL1 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED.
0b1	PL0 and PL1 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

CP15BEN is optional, but if it is implemented in the SCTLR then it must also be implemented in the SCTLR_EL1, SCTLR_EL2, and HSCTLR.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

When an implementation does not implement CP15BEN, access to this field is **RAO/WI**.

LSMAOE, bit [4]

When FEAT_LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

LSMAOE	Meaning
0b0	For all memory accesses at EL1 or EL0, T32 and A32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1	The ordering and interrupt behavior of T32 and A32 Load Multiple and Store Multiple at EL1 or EL0 is as defined for Armv8.0.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

Otherwise:

Reserved, RES1.

nTLSMD, bit [3]

When FEAT_LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

nTLSMD	Meaning
0b0	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL1 or EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/ Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL1 or EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/ Device-nGnRnE memory are not trapped.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

Otherwise:

Reserved, RES1.

C, bit [2]

Cacheability control, for data accesses at EL1 and EL0:

C	Meaning
0b0	All data access to Normal memory from PL1 and PL0, and all accesses to the PL1&0 stage 1 translation tables, are Non-cacheable for all levels of data and unified cache.
0b1	All data access to Normal memory from PL1 and PL0, and all accesses to the PL1&0 stage 1 translation tables, can be cached at all levels of data and unified cache.

The PE ignores SCTLR.C, and data accesses to Normal memory from EL1 and EL0 are Cacheable, if either:

- EL2 is using AArch32 and the value of HCR.DC is 1.
- EL2 is using AArch64 and the value of HCR_EL2.DC is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at PL1 and PL0:

A	Meaning
0b0	Alignment fault checking disabled when executing at PL1 or PL0. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.
0b1	Alignment fault checking enabled when executing at PL1 or PL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

M, bit [0]

MMU enable for EL1 and EL0 stage 1 address translation. Possible values of this bit are:

M	Meaning
0b0	EL1 and EL0 stage 1 address translation disabled. See the SCTLR.I field for the behavior of instruction accesses to Normal memory.
0b1	EL1 and EL0 stage 1 address translation enabled.

The PE behaves as if the value of the SCTLR.M field is 0 for all purposes other than returning the value of a direct read of the field if either:

- EL2 is using AArch32 and the value of HCR.{DC, TGE} is not {0, 0}.
- EL2 is using AArch64 and the value of HCR_EL2.{DC, TGE} is not {0, 0}.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing SCTLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = SCTLR_NS;
    else
        R[t] = SCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = SCTLR_NS;
    else
        R[t] = SCTLR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = SCTLR_S;
    else
        R[t] = SCTLR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        SCTLR_NS = R[t];
    else
        SCTLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        SCTLR_NS = R[t];
    else
        SCTLR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            SCTLR_S = R[t];
        else
            SCTLR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SDCR, Secure Debug Control Register

The SDCR characteristics are:

Purpose

Provides EL3 configuration options for self-hosted debug, trace, and the Performance Monitors Extension.

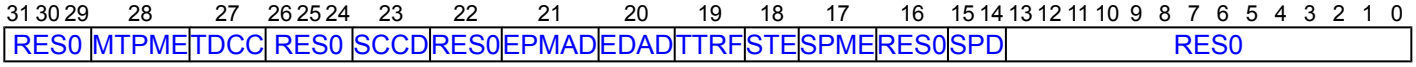
Configuration

This register is present only when FEAT_AA32EL3 is implemented. Otherwise, direct accesses to SDCR are UNDEFINED.

Attributes

SDCR is a 32-bit register.

Field descriptions



Bits [31:29]

Reserved, RES0.

MTPME, bit [28]

When FEAT_MTPMU is implemented:

Multi-threaded PMU Enable. Enables use of the PMEVTYPER<n>.MT bits.

MTPME	Meaning
0b0	FEAT_MTPMU is disabled. The Effective value of PMEVTYPER<n>.MT is 0.
0b1	PMEVTYPER<n>.MT bits not affected by this field.

If FEAT_MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this field is 0.

The reset behavior of this field is:

- On a Cold reset, when the highest implemented Exception level is EL3, this field resets to '1'.

Otherwise:

Reserved, RES0.

TDCC, bit [27]

When FEAT_FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel in modes other than Monitor mode to Monitor mode.

TDCC	Meaning
0b0	This control does not cause any register accesses to be trapped.
0b1	Accesses to the DCC registers in modes other than Monitor mode generate a Monitor Trap exception, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state.

The DCC registers trapped by this control are:

- DBGDTRRXext, DBGDTRTXext, DBGDSCRint, DBGDCCINT, and, when the PE is in Non-debug state, DBGDTRRXint and DBGDTRTXint.

When the PE is in Debug state, SDCR.TDCC does not trap any accesses to:

- DBGDTRRXint and DBGDTRTXint.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [26:24]

Reserved, RES0.

SCCD, bit [23]

When FEAT_PMUv3p5 is implemented:

Secure Cycle Counter Disable. Prohibits PMCCNTR from counting in Secure state and EL3.

SCCD	Meaning
0b0	Cycle counting by PMCCNTR is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR is prohibited in Secure state and EL3.

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [22]

Reserved, RES0.

EPMAD, bit [21]

When FEAT_Debugv8p4 is implemented and FEAT_PMUv3_EXT is implemented:

External Performance Monitors Non-secure access disable. Controls Non-secure access to Performance Monitors registers by an external debugger.

EPMAD	Meaning
0b0	No accesses from an external debugger to the Performance Monitor registers are prohibited by this control.
0b1	Non-secure accesses from an external debugger to the affected Performance Monitor registers are prohibited.

Otherwise, if EL3 is not implemented and the Effective value of SCR.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

When FEAT_PMUv3_EXT is implemented:

External Performance Monitors access disable. Controls access to Performance Monitors registers by an external debugger.

EPMAD	Meaning
0b0	No accesses from an external debugger to the Performance Monitor registers are prohibited by this control.
0b1	If the IMPLEMENTATION DEFINED authentication interface function ExternalSecureInvasiveDebugEnabled() returns FALSE, then accesses from an external debugger to the affected Performance Monitor registers are prohibited.

Otherwise, if EL3 is not implemented and the Effective value of SCR.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

EDAD, bit [20]
When FEAT_Debugv8p4 is implemented:

External debug Non-secure access disable. Controls Non-secure access to breakpoint, watchpoint, and OSLAR_EL1 registers by an external debugger.

EDAD	Meaning
0b0	No accesses from an external debugger to the debug registers are prohibited by this control.
0b1	Non-secure accesses from an external debugger to the affected debug registers are prohibited.

If EL3 is not implemented and the Effective value of SCR.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

When FEAT_Debugv8p2 is implemented:

External debug access disable. Controls access to breakpoint, watchpoint, and OSLAR_EL1 registers by an external debugger.

EDAD	Meaning
0b0	No accesses from an external debugger to the debug registers are prohibited by this control.
0b1	If the IMPLEMENTATION DEFINED authentication interface function `ExternalSecureInvasiveDebugEnabled()` returns FALSE, then accesses from an external debugger to the affected debug registers are prohibited.

If EL3 is not implemented and the Effective value of SCR.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to `'0'`.

Otherwise:

External debug access disable. Controls access to breakpoint, watchpoint, and optionally OSLAR_EL1 registers by an external debugger.

EDAD	Meaning
0b0	No accesses from an external debugger to the debug registers are prohibited by this control.
0b1	If the IMPLEMENTATION DEFINED authentication interface function `ExternalSecureInvasiveDebugEnabled()` returns FALSE, then accesses from an external debugger to the affected debug registers are prohibited.

It is IMPLEMENTATION DEFINED whether accesses to OSLAR_EL1 from an external debugger are affected by this control.

If EL3 is not implemented and the Effective value of SCR.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to `'0'`.

TTRF, bit [19]

When FEAT_TRF is implemented:

Trap Trace Filter controls. Controls whether accesses in modes other than Monitor mode to the trace filter control registers generate a Monitor Trap exception.

TTRF	Meaning
0b0	Accesses to HTRFCR and TRFCR are not affected by this control bit.
0b1	When not in Monitor mode, accesses to HTRFCR and TRFCR generate a Monitor Trap exception, unless the access generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

STE, bit [18]

When FEAT_TRF is implemented:

Secure Trace Enable. This field enables tracing in Secure state and controls the level of authentication required by an external debugger to enable external tracing.

STE	Meaning
0b0	Trace is prohibited in Secure state unless overridden by the IMPLEMENTATION DEFINED authentication interface.
0b1	Trace in Secure state is not affected by this field.

This field also controls the level of authentication required by an external debugger to enable external tracing. See 'Register controls to enable self-hosted trace'.

If EL3 is not implemented and the Effective value of SCR.NS is 0, the PE behaves as if this field is set to 1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

SPME, bit [17]
When FEAT_PMUv3 is implemented and FEAT_Debugv8p2 is implemented:

Secure Performance Monitors Enable. Controls event counting in Secure state.

SPME	Meaning
0b0	Event counting is prohibited in Secure state. If PMCR.DP is 1, PMCCNTR is disabled in Secure state. Otherwise, PMCCNTR is not affected by this mechanism.
0b1	Event counting and PMCCNTR are not affected by this mechanism.

This field affects the operation of all event counters in Secure state, and if PMCR.DP is 1, the operation of PMCCNTR in Secure state. When PMCR.DP is 0, PMCCNTR is not affected by this field.

If EL3 is not implemented and the Effective value of SCR.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

When FEAT_PMUv3 is implemented:

Secure Performance Monitors Enable. Controls event counting in Secure state.

SPME	Meaning
0b0	If ExternalSecureNoninvasiveDebugEnabled() is FALSE, event counting is prohibited in Secure state, and if PMCR.DP is 1, PMCCNTR is disabled in Secure state.
0b1	Event counting and PMCCNTR are not affected by this mechanism.

If ExternalSecureNoninvasiveDebugEnabled() is TRUE, the event counters and PMCCNTR are not affected by this field.

Otherwise, this field affects the operation of all event counters in Secure state, and if PMCR.DP is 1, the operation of PMCCNTR in Secure state. When PMCR.DP is 0, PMCCNTR is not affected by this field.

If EL3 is not implemented and the Effective value of SCR.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [16]

Reserved, RES0.

SPD, bits [15:14]

AArch32 Secure self-hosted Privileged Debug. Enables or disables debug exceptions from EL3, other than Breakpoint Instruction exceptions.

SPD	Meaning
0b00	Legacy mode. Debug exceptions from EL3 are enabled by the authentication interface.
0b10	Secure privileged debug disabled. Debug exceptions from EL3 are disabled.
0b11	Secure privileged debug enabled. Debug exceptions from EL3 are enabled.

Other values are reserved, and have the CONSTRAINED UNPREDICTABLE behavior that they must have the same behavior as 0b00. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

This field has no effect on Breakpoint Instruction exceptions. These are always enabled.

This field is ignored in Non-secure state.

If debug exceptions from EL3 are enabled, then debug exceptions from Secure EL0 are also enabled.

Otherwise, debug exceptions from Secure EL0 are enabled only if the value of SDER.SUIDEN is 1.

If EL3 is not implemented and the Effective value of SCR.NS is 0, then the Effective value of this field is 0b11.

The reset behavior of this field is:

- On a Warm reset, when the highest implemented Exception level is EL3, this field resets to '00'.

Bits [13:0]

Reserved, RES0.

Accessing SDCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL3) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    R[t] = SDCR;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL3) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
IsCurrentSecurityState(SS_Secure) then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        SDCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SDER, Secure Debug Enable Register

The SDER characteristics are:

Purpose

Controls invasive and non-invasive debug in the Secure EL0 mode.

Configuration

AArch32 System register SDER bits [31:0] are architecturally mapped to AArch64 System register SDER32_EL2[31:0] when EL2 is implemented and FEAT_SEL2 is implemented.

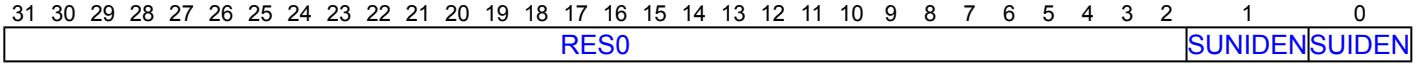
AArch32 System register SDER bits [31:0] are architecturally mapped to AArch64 System register SDER32_EL3[31:0] when EL3 is implemented.

This register is present only when (EL3 is implemented and FEAT_AA32EL3 is implemented) or (FEAT_AA32EL1 is implemented and Secure EL1 is implemented). Otherwise, direct accesses to SDER are UNDEFINED.

Attributes

SDER is a 32-bit register.

Field descriptions



Bits [31:2]

Reserved, RES0.

SUNIDEN, bit [1]

Secure User Non-Invasive Debug Enable.

SUNIDEN	Meaning
0b0	This bit has no effect on non-invasive debug.
0b1	Non-invasive debug is allowed in Secure EL0 using AArch32.

When EL3 or Secure EL1 is using AArch32, the forms of non-invasive debug affected by this control are:

- The PC Sample-based Profiling Extension. See About the PC Sample-based Profiling Extension.
- When SelfHostedTraceEnabled() == FALSE, processor trace.
- When EL3 is implemented, Performance Monitors.

When Secure EL1 is using AArch64, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

SUIDEN, bit [0]

When EL3 is implemented:

Secure User Invasive Debug Enable.

SUIDEN	Meaning
0b0	This bit does not affect the generation of debug exceptions at Secure EL0.
0b1	If EL3 or EL1 is using AArch32, debug exceptions from Secure EL0 are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Accessing SDER

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0001	0b001

```
if !((HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3)) || (IsFeatureImplemented(FEAT_AA32EL1) &&
HaveELUsingSecurityState(EL1, TRUE))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        R[t] = SDER;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    R[t] = SDER;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0001	0b001

```
if !((HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3)) || (IsFeatureImplemented(FEAT_AA32EL1) &&
HaveELUsingSecurityState(EL1, TRUE))) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T1
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        SDER = R[t];
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        SDER = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR, Saved Program Status Register

The SPSR characteristics are:

Purpose

Holds the saved process state for the current mode.

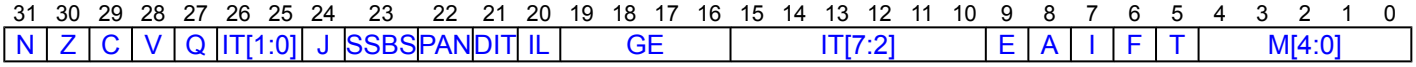
Configuration

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to SPSR are UNDEFINED.

Attributes

SPSR is a 32-bit register.

Field descriptions



N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to the current mode, and copied to PSTATE.N on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to the current mode, and copied to PSTATE.Z on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to the current mode, and copied to PSTATE.C on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to the current mode, and copied to PSTATE.V on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to the current mode, and copied to PSTATE.Q on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to the current mode, and copied to PSTATE.IT on executing an exception return operation in the current mode.

On executing an exception return operation in the current mode, SPSR.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR[26:25].
- IT[7:2] is SPSR[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to the current mode, and copied to PSTATE.SSBS on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to the current mode, and copied to PSTATE.PAN on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to the current mode, and copied to PSTATE.DIT on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to the current mode, and copied to PSTATE.IL on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to the current mode, and copied to PSTATE.GE on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to the current mode, and copied to PSTATE.E on executing an exception return operation in the current mode.

If the implementation does not support big-endian operation, SPSR.E is RES0. If the implementation does not support little-endian operation, SPSR.E is RES1. On executing an exception return operation in the current mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to the current mode, and copied to PSTATE.A on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to the current mode, and copied to PSTATE.I on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to the current mode, and copied to PSTATE.F on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to the current mode, and copied to PSTATE.T on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to the current mode, and copied to PSTATE.M[4:0] on executing an exception return operation in the current mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10110	Monitor.
0b10111	Abort.
0b11010	Hyp.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in the current mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR

SPSR can be read using the MRS instruction and written using the MSR (register) or MSR (immediate) instructions.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_abt, Saved Program Status Register (Abort mode)

The SPSR_abt characteristics are:

Purpose

Holds the saved process state when an exception is taken to Abort mode.

Configuration

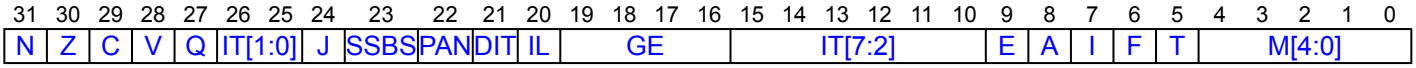
AArch32 System register SPSR_abt bits [31:0] are architecturally mapped to AArch64 System register SPSR_abt[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to SPSR_abt are UNDEFINED.

Attributes

SPSR_abt is a 32-bit register.

Field descriptions



N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Abort mode, and copied to PSTATE.N on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Abort mode, and copied to PSTATE.Z on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Abort mode, and copied to PSTATE.C on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Abort mode, and copied to PSTATE.V on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Abort mode, and copied to PSTATE.Q on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Abort mode, and copied to PSTATE.IT on executing an exception return operation in Abort mode.

On executing an exception return operation in Abort mode, SPSR_abt.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_abt[26:25].
- IT[7:2] is SPSR_abt[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]
When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Abort mode, and copied to PSTATE.SSBS on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]
When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Abort mode, and copied to PSTATE.PAN on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Abort mode, and copied to PSTATE.DIT on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Abort mode, and copied to PSTATE.IL on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Abort mode, and copied to PSTATE.GE on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Abort mode, and copied to PSTATE.E on executing an exception return operation in Abort mode.

If the implementation does not support big-endian operation, SPSR_abt.E is RES0. If the implementation does not support little-endian operation, SPSR_abt.E is RES1. On executing an exception return operation in Abort mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_abt.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_abt.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to Abort mode, and copied to PSTATE.A on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Abort mode, and copied to PSTATE.I on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Abort mode, and copied to PSTATE.F on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Abort mode, and copied to PSTATE.T on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Abort mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Abort mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10111	Abort.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_abt.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Abort mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_abt

SPSR_abt is accessible in all modes other than User mode and Abort mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS{<c>}{<q>} <Rd>, SPSR_abt
```

R	M	M1
0b1	0b1	0b0100

```
MSR{<c>}{<q>} SPSR_abt, <Rn>
```

R	M	M1

0b1	0b1	0b0100

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_fiq, Saved Program Status Register (FIQ mode)

The SPSR_fiq characteristics are:

Purpose

Holds the saved process state when an exception is taken to FIQ mode.

Configuration

AArch32 System register SPSR_fiq bits [31:0] are architecturally mapped to AArch64 System register SPSR_fiq[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to SPSR_fiq are UNDEFINED.

Attributes

SPSR_fiq is a 32-bit register.

Field descriptions

31	30	29	28	27	26 25	24	23 22	21	20	19 18 17 16	15 14 13 12 11 10	9	8	7	6	5	4 3 2 1 0
N	Z	C	V	Q	IT[1:0]	J	SSBS	PAN	DIT	IL	GE	IT[7:2]		E	A	I	F	T	M[4:0]

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to FIQ mode, and copied to PSTATE.N on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to FIQ mode, and copied to PSTATE.Z on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to FIQ mode, and copied to PSTATE.C on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to FIQ mode, and copied to PSTATE.V on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to FIQ mode, and copied to PSTATE.Q on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to FIQ mode, and copied to PSTATE.IT on executing an exception return operation in FIQ mode.

On executing an exception return operation in FIQ mode, SPSR_fiq.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_fiq[26:25].
- IT[7:2] is SPSR_fiq[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to FIQ mode, and copied to PSTATE.SSBS on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to FIQ mode, and copied to PSTATE.PAN on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to FIQ mode, and copied to PSTATE.DIT on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to FIQ mode, and copied to PSTATE.IL on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to FIQ mode, and copied to PSTATE.GE on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to FIQ mode, and copied to PSTATE.E on executing an exception return operation in FIQ mode.

If the implementation does not support big-endian operation, SPSR_fiq.E is RES0. If the implementation does not support little-endian operation, SPSR_fiq.E is RES1. On executing an exception return operation in FIQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_fiq.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_fiq.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to FIQ mode, and copied to PSTATE.A on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to FIQ mode, and copied to PSTATE.I on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to FIQ mode, and copied to PSTATE.F on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to FIQ mode, and copied to PSTATE.T on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to FIQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in FIQ mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10111	Abort.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_fiq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in FIQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_fiq

SPSR_fiq is accessible in all modes other than User mode and FIQ mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS{<c>}{<q>} <Rd>, SPSR_fiq
```

R	M	M1
0b1	0b0	0b1110

```
MSR{<c>}{<q>} SPSR_fiq, <Rn>
```

R	M	M1

0b1	0b0	0b1110

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_hyp, Saved Program Status Register (Hyp mode)

The SPSR_hyp characteristics are:

Purpose

Holds the saved process state when an exception is taken to Hyp mode.

Configuration

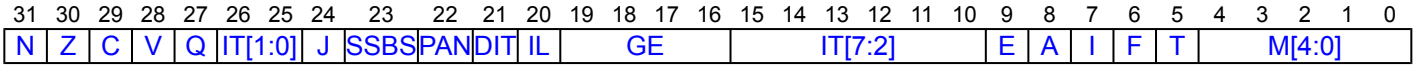
AArch32 System register SPSR_hyp bits [31:0] are architecturally mapped to AArch64 System register SPSR_EL2[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to SPSR_hyp are UNDEFINED.

Attributes

SPSR_hyp is a 32-bit register.

Field descriptions



N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Hyp mode, and copied to PSTATE.N on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Hyp mode, and copied to PSTATE.Z on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Hyp mode, and copied to PSTATE.C on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Hyp mode, and copied to PSTATE.V on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Hyp mode, and copied to PSTATE.Q on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Hyp mode, and copied to PSTATE.IT on executing an exception return operation in Hyp mode.

On executing an exception return operation in Hyp mode, SPSR_hyp.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_hyp[26:25].
- IT[7:2] is SPSR_hyp[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Hyp mode, and copied to PSTATE.SSBS on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Hyp mode, and copied to PSTATE.PAN on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Hyp mode, and copied to PSTATE.DIT on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Hyp mode, and copied to PSTATE.IL on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Hyp mode, and copied to PSTATE.GE on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Hyp mode, and copied to PSTATE.E on executing an exception return operation in Hyp mode.

If the implementation does not support big-endian operation, SPSR_hyp.E is RES0. If the implementation does not support little-endian operation, SPSR_hyp.E is RES1. On executing an exception return operation in Hyp mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_hyp.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_hyp.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to Hyp mode, and copied to PSTATE.A on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Hyp mode, and copied to PSTATE.I on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Hyp mode, and copied to PSTATE.F on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Hyp mode, and copied to PSTATE.T on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Hyp mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Hyp mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10111	Abort.
0b11010	Hyp.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_hyp.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Hyp mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_hyp

SPSR_hyp is accessible only in Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS{<c>}{<q>} <Rd>, SPSR_hyp
```

R	M	M1
0b1	0b1	0b1110

```
MSR{<c>}{<q>} SPSR_hyp, <Rn>
```

R	M	M1
0b1	0b1	0b1110

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_irq, Saved Program Status Register (IRQ mode)

The SPSR_irq characteristics are:

Purpose

Holds the saved process state when an exception is taken to IRQ mode.

Configuration

AArch32 System register SPSR_irq bits [31:0] are architecturally mapped to AArch64 System register SPSR_irq[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to SPSR_irq are UNDEFINED.

Attributes

SPSR_irq is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N	Z	C	V	Q	IT[1:0]		J	SSBS	PAN	DIT	IL	GE				IT[7:2]						E	A	I	F	T	M[4:0]				

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to IRQ mode, and copied to PSTATE.N on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to IRQ mode, and copied to PSTATE.Z on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to IRQ mode, and copied to PSTATE.C on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to IRQ mode, and copied to PSTATE.V on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to IRQ mode, and copied to PSTATE.Q on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to IRQ mode, and copied to PSTATE.IT on executing an exception return operation in IRQ mode.

On executing an exception return operation in IRQ mode, SPSR_irq.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_irq[26:25].
- IT[7:2] is SPSR_irq[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]
When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to IRQ mode, and copied to PSTATE.SSBS on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]
When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to IRQ mode, and copied to PSTATE.PAN on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]
When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to IRQ mode, and copied to PSTATE.DIT on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to IRQ mode, and copied to PSTATE.IL on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to IRQ mode, and copied to PSTATE.GE on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to IRQ mode, and copied to PSTATE.E on executing an exception return operation in IRQ mode.

If the implementation does not support big-endian operation, SPSR_irq.E is RES0. If the implementation does not support little-endian operation, SPSR_irq.E is RES1. On executing an exception return operation in IRQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_irq.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_irq.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to IRQ mode, and copied to PSTATE.A on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to IRQ mode, and copied to PSTATE.I on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to IRQ mode, and copied to PSTATE.F on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to IRQ mode, and copied to PSTATE.T on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to IRQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in IRQ mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10111	Abort.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_irq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in IRQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_irq

SPSR_irq is accessible in all modes other than User mode and IRQ mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS{<c>}{<q>} <Rd>, SPSR_irq
```

R	M	M1
0b1	0b1	0b0000

```
MSR{<c>}{<q>} SPSR_irq, <Rn>
```

R	M	M1

0b1	0b1	0b0000

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_mon, Saved Program Status Register (Monitor mode)

The SPSR_mon characteristics are:

Purpose

Holds the saved process state when an exception is taken to Monitor mode.

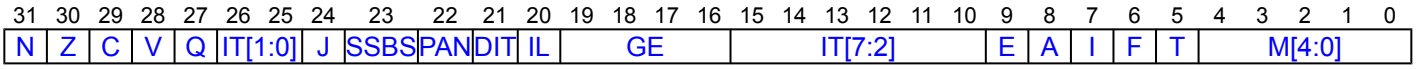
Configuration

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to SPSR_mon are UNDEFINED.

Attributes

SPSR_mon is a 32-bit register.

Field descriptions



N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Monitor mode, and copied to PSTATE.N on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Monitor mode, and copied to PSTATE.Z on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Monitor mode, and copied to PSTATE.C on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Monitor mode, and copied to PSTATE.V on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Monitor mode, and copied to PSTATE.Q on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Monitor mode, and copied to PSTATE.IT on executing an exception return operation in Monitor mode.

On executing an exception return operation in Monitor mode, SPSR_mon.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_mon[26:25].
- IT[7:2] is SPSR_mon[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Monitor mode, and copied to PSTATE.SSBS on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Monitor mode, and copied to PSTATE.PAN on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Monitor mode, and copied to PSTATE.DIT on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Monitor mode, and copied to PSTATE.IL on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Monitor mode, and copied to PSTATE.GE on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Monitor mode, and copied to PSTATE.E on executing an exception return operation in Monitor mode.

If the implementation does not support big-endian operation, SPSR_mon.E is RES0. If the implementation does not support little-endian operation, SPSR_mon.E is RES1. On executing an exception return operation in Monitor mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_mon.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_mon.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to Monitor mode, and copied to PSTATE.A on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Monitor mode, and copied to PSTATE.I on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Monitor mode, and copied to PSTATE.F on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Monitor mode, and copied to PSTATE.T on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Monitor mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Monitor mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10110	Monitor.
0b10111	Abort.
0b11010	Hyp.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_mon.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Monitor mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_mon

SPSR_mon is only accessible in EL3 modes other than Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS{<c>}{<q>} <Rd>, SPSR_mon
```

R	M	M1
0b1	0b1	0b1100

```
MSR{<c>}{<q>} SPSR_mon, <Rn>
```

R	M	M1
0b1	0b1	0b1100

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_svc, Saved Program Status Register (Supervisor mode)

The SPSR_svc characteristics are:

Purpose

Holds the saved process state when an exception is taken to Supervisor mode.

Configuration

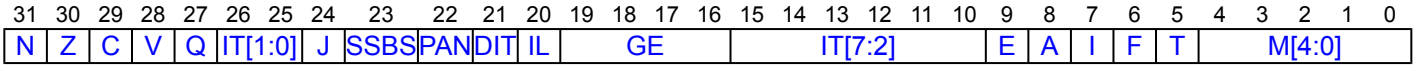
AArch32 System register SPSR_svc bits [31:0] are architecturally mapped to AArch64 System register SPSR_EL1[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to SPSR_svc are UNDEFINED.

Attributes

SPSR_svc is a 32-bit register.

Field descriptions



N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Supervisor mode, and copied to PSTATE.N on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Supervisor mode, and copied to PSTATE.Z on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Supervisor mode, and copied to PSTATE.C on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Supervisor mode, and copied to PSTATE.V on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Supervisor mode, and copied to PSTATE.Q on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Supervisor mode, and copied to PSTATE.IT on executing an exception return operation in Supervisor mode.

On executing an exception return operation in Supervisor mode, SPSR_svc.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_svc[26:25].
- IT[7:2] is SPSR_svc[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Supervisor mode, and copied to PSTATE.SSBS on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Supervisor mode, and copied to PSTATE.PAN on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Supervisor mode, and copied to PSTATE.DIT on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Supervisor mode, and copied to PSTATE.IL on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Supervisor mode, and copied to PSTATE.GE on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Supervisor mode, and copied to PSTATE.E on executing an exception return operation in Supervisor mode.

If the implementation does not support big-endian operation, SPSR_svc.E is RES0. If the implementation does not support little-endian operation, SPSR_svc.E is RES1. On executing an exception return operation in Supervisor mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_svc.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_svc.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to Supervisor mode, and copied to PSTATE.A on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Supervisor mode, and copied to PSTATE.I on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Supervisor mode, and copied to PSTATE.F on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Supervisor mode, and copied to PSTATE.T on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Supervisor mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Supervisor mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10111	Abort.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_svc.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Supervisor mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_svc

SPSR_svc is accessible in all modes other than User mode and Supervisor mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS{<c>}{<q>} <Rd>, SPSR_svc
```

R	M	M1
0b1	0b1	0b0010

```
MSR{<c>}{<q>} SPSR_svc, <Rn>
```

R	M	M1

0b1	0b1	0b0010

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_und, Saved Program Status Register (Undefined mode)

The SPSR_und characteristics are:

Purpose

Holds the saved process state when an exception is taken to Undefined mode.

Configuration

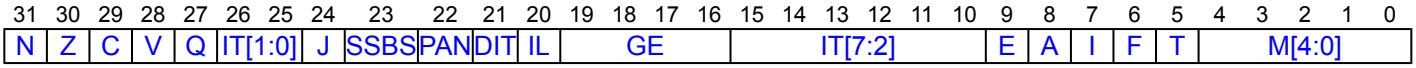
AArch32 System register SPSR_und bits [31:0] are architecturally mapped to AArch64 System register SPSR_und[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to SPSR_und are UNDEFINED.

Attributes

SPSR_und is a 32-bit register.

Field descriptions



N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Undefined mode, and copied to PSTATE.N on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Undefined mode, and copied to PSTATE.Z on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Undefined mode, and copied to PSTATE.C on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Undefined mode, and copied to PSTATE.V on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Undefined mode, and copied to PSTATE.Q on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Undefined mode, and copied to PSTATE.IT on executing an exception return operation in Undefined mode.

On executing an exception return operation in Undefined mode, SPSR_und.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_und[26:25].
- IT[7:2] is SPSR_und[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Undefined mode, and copied to PSTATE.SSBS on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Undefined mode, and copied to PSTATE.PAN on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]
When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Undefined mode, and copied to PSTATE.DIT on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Undefined mode, and copied to PSTATE.IL on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Undefined mode, and copied to PSTATE.GE on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Undefined mode, and copied to PSTATE.E on executing an exception return operation in Undefined mode.

If the implementation does not support big-endian operation, SPSR_und.E is RES0. If the implementation does not support little-endian operation, SPSR_und.E is RES1. On executing an exception return operation in Undefined mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_und.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_und.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on taking an exception to Undefined mode, and copied to PSTATE.A on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Undefined mode, and copied to PSTATE.I on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Undefined mode, and copied to PSTATE.F on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Undefined mode, and copied to PSTATE.T on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Undefined mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Undefined mode.

M[4:0]	Meaning
0b10000	User.
0b10001	FIQ.
0b10010	IRQ.
0b10011	Supervisor.
0b10111	Abort.
0b11011	Undefined.
0b11111	System.

Other values are reserved. If SPSR_und.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Undefined mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_und

SPSR_und is accessible in all modes other than User mode and Undefined mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS{<c>}{<q>} <Rd>, SPSR_und
```

R	M	M1
0b1	0b1	0b0110

```
MSR{<c>}{<q>} SPSR_und, <Rn>
```

R	M	M1

0b1	0b1	0b0110

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCMTR, TCM Type Register

The TCMTR characteristics are:

Purpose

Provides information about the implementation of the TCM.

Configuration

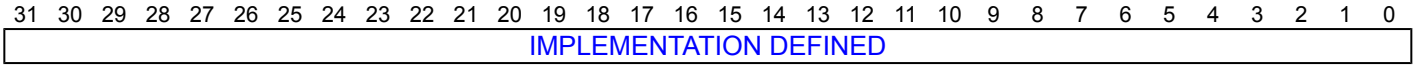
This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TCMTR are UNDEFINED.

If EL1 or above can use AArch32 then this register must be implemented.

Attributes

TCMTR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing TCMTR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = TCMTR;
elsif PSTATE.EL == EL2 then
    R[t] = TCMTR;
elsif PSTATE.EL == EL3 then
    R[t] = TCMTR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIALL, TLB Invalidate All

The TLBIALL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
  - Would be required for the EL1&0 translation regime.
  - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2, and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIALL are UNDEFINED.

Attributes

TLBIALL is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing TLBIALL

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HCR_EL2.FB
== '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_ExcludeXS);
        else
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_AllAttr);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FB ==
'1' then
        AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_AllAttr);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS);
        else
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL30, Broadcast_NSH, TLBI_ExcludeXS);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIALLH, TLB Invalidate All, Hyp mode

The TLBIALLH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIALLH are UNDEFINED.

Attributes

TLBIALLH is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing TLBIALLH

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_NSH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_ALL(SS_NonSecure, Regime_EL2, Broadcast_NSH, TLBI_AllAttr);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIALLHIS, TLB Invalidate All, Hyp mode, Inner Shareable

The TLBIALLHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIALLHIS are UNDEFINED.

Attributes

TLBIALLHIS is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing TLBIALLHIS

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Broadcast_ISH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_ALL(SS_NonSecure, Regime_EL2, Broadcast_ISH, TLBI_AllAttr);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIALLIS, TLB Invalidate All, Inner Shareable

The TLBIALLIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
  - Would be required for the EL1&0 translation regime.
  - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2, and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the PL1&0 translation regime and matches the current VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIALLIS are UNDEFINED.

Attributes

TLBIALLIS is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing TLBIALLIS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TTLBIS
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS);
        else
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL30, Broadcast_ISH, TLBI_ExcludeXS);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIALLNSNH, TLB Invalidate All, Non-Secure Non-Hyp

The TLBIALLNSNH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for stage 1 or stage 2 of the Non-secure PL1&0 translation regime, regardless of the associated VMID.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIALLNSNH are UNDEFINED.

Attributes

TLBIALLNSNH is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing TLBIALLNSNH

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0111	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_NSH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_ALL(SS_NonSecure, Regime_EL10, Broadcast_NSH, TLBI_AllAttr);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIALLNSNHIS, TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable

The TLBIALLNSNHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for stage 1 or stage 2 of the Non-secure PL1&0 translation regime, regardless of the associated VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIALLNSNHIS are UNDEFINED.

Attributes

TLBIALLNSNHIS is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing TLBIALLNSNHIS

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Broadcast_ISH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_ALL(SS_NonSecure, Regime_EL10, Broadcast_ISH, TLBI_AllAttr);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIASID, TLB Invalidate by ASID match

The TLBIASID characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

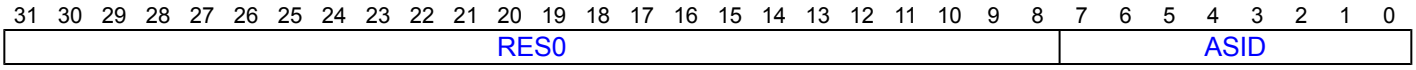
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIASID are UNDEFINED.

Attributes

TLBIASID is a 32-bit System instruction.

Field descriptions



Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

Executing TLBIASID

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0111	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HCR_EL2.FB
== '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_AllAttr, R[t]);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FB ==
'1' then
        AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBI_AllAttr, R[t]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBI_AllAttr,
R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_ASID(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_NSH, TLBI_AllAttr,
R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIASIDIS, TLB Invalidate by ASID match, Inner Shareable

The TLBIASIDIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

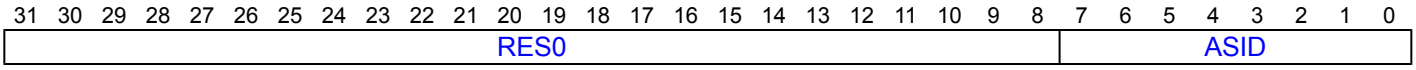
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIASIDIS are UNDEFINED.

Attributes

TLBIASIDIS is a 32-bit System instruction.

Field descriptions



Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

Executing TLBIASIDIS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TTLBIS
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBI_AllAttr,
R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_ASID(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_ISH, TLBI_AllAttr,
R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIIPAS2, TLB Invalidate by Intermediate Physical Address, Stage 2

The TLBIIPAS2 characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- SCR.NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIIPAS2 are UNDEFINED.

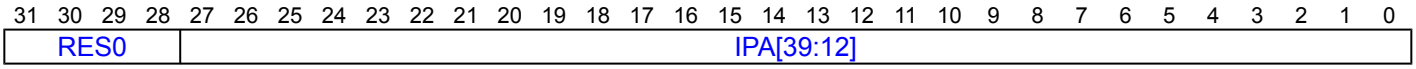
Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2 is a 32-bit System instruction.

Field descriptions



Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing TLBIIPAS2

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        return;
    else
        AArch32.TLBI_IPAS2(SS_NonSecure, Regime_EL10, VMID_NONE, Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIIPAS2IS, TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable

The TLBIIPAS2IS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- SCR.NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIIPAS2IS are UNDEFINED.

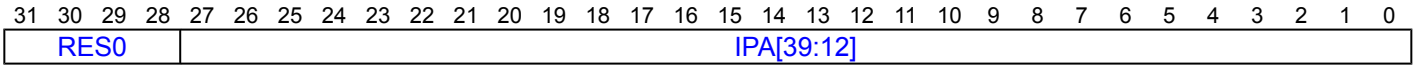
Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2IS is a 32-bit System instruction.

Field descriptions



Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing TLBIIPAS2IS

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        return;
    else
        AArch32.TLBI_IPAS2(SS_NonSecure, Regime_EL10, VMID_NONE, Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIIPAS2L, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level

The TLBIIPAS2L characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- SCR.NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIIPAS2L are UNDEFINED.

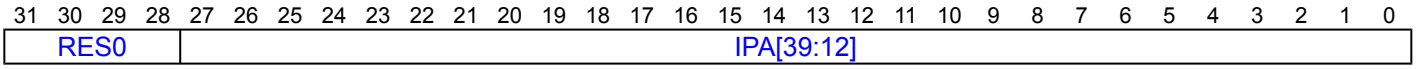
Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2L is a 32-bit System instruction.

Field descriptions



Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing TLBIIPAS2L

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        return;
    else
        AArch32.TLBI_IPAS2(SS_NonSecure, Regime_EL10, VMID_NONE, Broadcast_NSH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIIPAS2LIS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable

The TLBIIPAS2LIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- SCR.NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIIPAS2LIS are UNDEFINED.

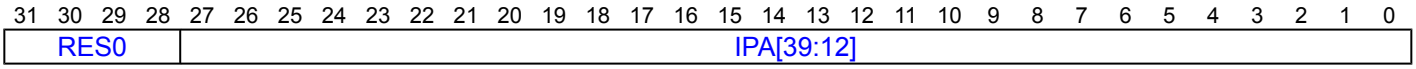
Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2LIS is a 32-bit System instruction.

Field descriptions



Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing TLBIIPAS2LIS

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        return;
    else
        AArch32.TLBI_IPAS2(SS_NonSecure, Regime_EL10, VMID_NONE, Broadcast_ISH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVA, TLB Invalidate by VA

The TLBIMVA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

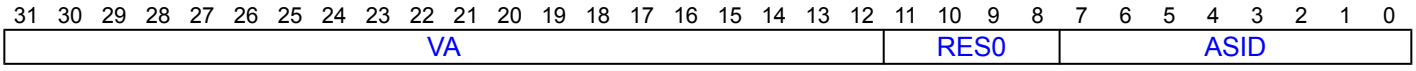
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIMVA are UNDEFINED.

Attributes

TLBIMVA is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

Executing TLBIMVA

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0111	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HCR_EL2.FB
== '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FB ==
'1' then
        AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAA, TLB Invalidate by VA, All ASID

The TLBIMVAA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

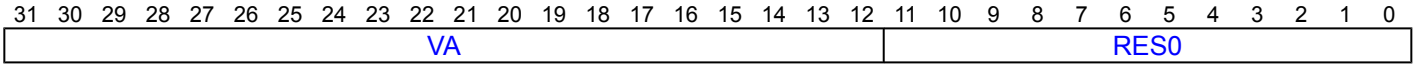
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIMVAA are UNDEFINED.

Attributes

TLBIMVAA is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing TLBIMVAA

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0111	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HCR_EL2.FB
== '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FB ==
'1' then
        AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_VAA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAAIS, TLB Invalidate by VA, All ASID, Inner Shareable

The TLBIMVAAIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

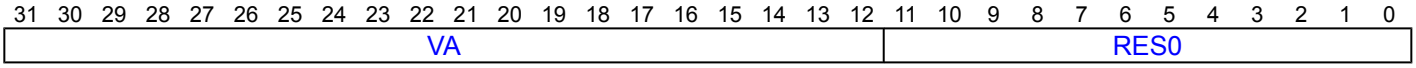
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIMVAAIS are UNDEFINED.

Attributes

TLBIMVAAIS is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing TLBIMVAAIS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0011	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TTLBIS
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_VAA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAAL, TLB Invalidate by VA, All ASID, Last level

The TLBIMVAAL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIMVAAL are UNDEFINED.

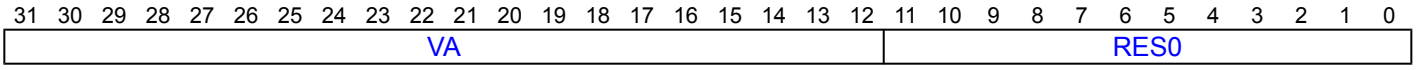
Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVAAL is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing TLBIMVAAL

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0111	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HCR_EL2.FB
== '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FB ==
'1' then
        AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_VAA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAALIS, TLB Invalidate by VA, All ASID, Last level, Inner Shareable

The TLBIMVAALIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIMVAALIS are UNDEFINED.

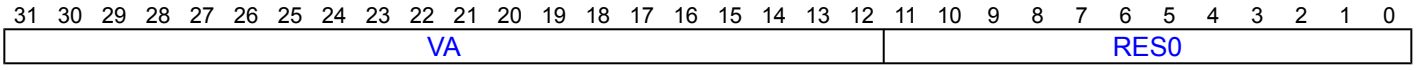
Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVAALIS is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing TLBIMVAALIS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b1000	0b0011	0b111

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TTLBIS
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_VAA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAH, TLB Invalidate by VA, Hyp mode

The TLBIMVAH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation only applies to the PE that executes this System instruction.

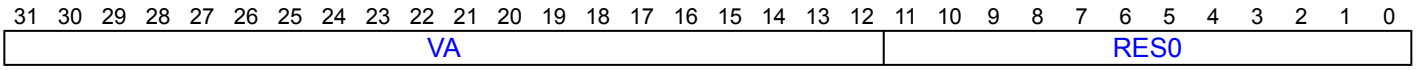
Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIMVAH are UNDEFINED.

Attributes

TLBIMVAH is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:0]

Reserved, RES0.

Executing TLBIMVAH

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0111	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_VA(SS_NonSecure, Regime_EL2, VMID[], Broadcast_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAHIS, TLB Invalidate by VA, Hyp mode, Inner Shareable

The TLBIMVAHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

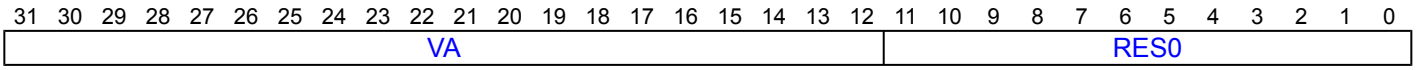
Configuration

This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIMVAHIS are UNDEFINED.

Attributes

TLBIMVAHIS is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:0]

Reserved, RES0.

Executing TLBIMVAHIS

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_VA(SS_NonSecure, Regime_EL2, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAIS, TLB Invalidate by VA, Inner Shareable

The TLBIMVAIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

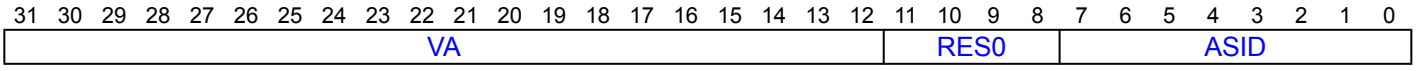
Configuration

This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIMVAIS are UNDEFINED.

Attributes

TLBIMVAIS is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

Executing TLBIMVAIS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TTLBIS
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_ISH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAL, TLB Invalidate by VA, Last level

The TLBIMVAL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

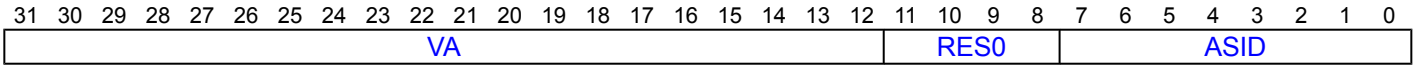
This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIMVAL are UNDEFINED.

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVAL is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

Executing TLBIMVAL

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0111	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HCR_EL2.FB
== '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.FB ==
'1' then
        AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ForcedISH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_NSH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_NSH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
```


Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVALH, TLB Invalidate by VA, Last level, Hyp mode

The TLBIMVALH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from the final level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation only applies to the PE that executes this System instruction.

Configuration

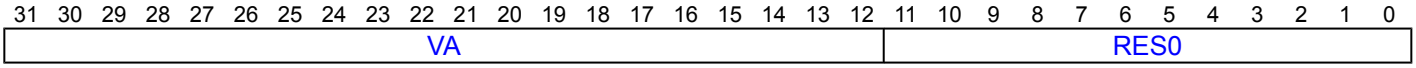
This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIMVALH are UNDEFINED.

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVALH is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:0]

Reserved, RES0.

Executing TLBIMVALH

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0111	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_NSH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_VA(SS_NonSecure, Regime_EL2, VMID[], Broadcast_NSH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVALHIS, TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable

The TLBIMVALHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from the final level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

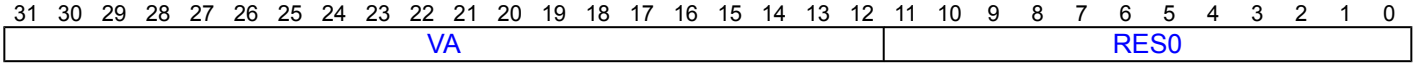
This instruction is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to TLBIMVALHIS are UNDEFINED.

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVALHIS is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:0]

Reserved, RES0.

Executing TLBIMVALHIS

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1000	0b0011	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Broadcast_ISH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_VA(SS_NonSecure, Regime_EL2, VMID[], Broadcast_ISH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVALIS, TLB Invalidate by VA, Last level, Inner Shareable

The TLBIMVALIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

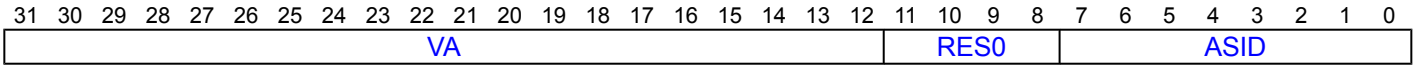
This instruction is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBIMVALIS are UNDEFINED.

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVALIS is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

Executing TLBIMVALIS

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1000	0b0011	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T8
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T8 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TTLB ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR2.TTLBIS
== '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_AA64EL2) &&
!ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled() && HCRX_EL2.FnXS ==
'1' then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Broadcast_ISH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Broadcast_ISH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBTR, TLB Type Register

The TLBTR characteristics are:

Purpose

Provides information about the TLB implementation. The register must define whether the implementation provides separate instruction and data TLBs, or a unified TLB. Normally, the IMPLEMENTATION DEFINED information in this register includes the number of lockable entries in the TLB.

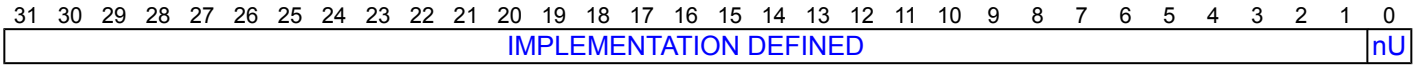
Configuration

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TLBTR are UNDEFINED.

Attributes

TLBTR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:1]

IMPLEMENTATION DEFINED.

nU, bit [0]

Not Unified TLB. Indicates whether the implementation has a unified TLB.

The value of this field is an IMPLEMENTATION DEFINED choice of:

nU	Meaning
0b0	Unified TLB.
0b1	Separate Instruction and Data TLBs.

Access to this field is **RO**.

Accessing TLBTR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TID1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        R[t] = TLBTR;
elsif PSTATE.EL == EL2 then
    R[t] = TLBTR;
elsif PSTATE.EL == EL3 then
    R[t] = TLBTR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDRPRW, PL1 Software Thread ID Register

The TPIDRPRW characteristics are:

Purpose

Provides a location where software executing at EL1 or higher can store thread identifying information that is not visible to software executing at EL0, for OS management purposes.

The PE makes no use of this register.

Configuration

This register is banked between TPIDRPRW and TPIDRPRW_S and TPIDRPRW_NS.

AArch32 System register TPIDRPRW bits [31:0] are architecturally mapped to AArch64 System register TPIDR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TPIDRPRW are UNDEFINED.

Note

The PE never updates this register.

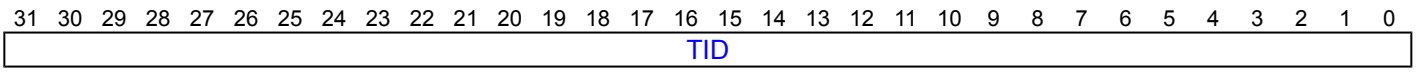
Attributes

TPIDRPRW is a 32-bit register.

This register has the following instances:

- TPIDRPRW, when EL3 is not implemented or FEAT_AA64 is implemented.
- TPIDRPRW_S, when FEAT_AA32EL3 is implemented.
- TPIDRPRW_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



TID, bits [31:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDRPRW

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TPIDRPRW_NS;
    else
        R[t] = TPIDRPRW;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TPIDRPRW_NS;
    else
        R[t] = TPIDRPRW;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = TPIDRPRW_S;
    else
        R[t] = TPIDRPRW_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TPIDRPRW_NS = R[t];
    else
        TPIDRPRW = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TPIDRPRW_NS = R[t];
    else
        TPIDRPRW = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        TPIDRPRW_S = R[t];
    else
        TPIDRPRW_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDRURO, PL0 Read-Only Software Thread ID Register

The TPIDRURO characteristics are:

Purpose

Provides a location where software executing at EL1 or higher can store thread identifying information that is visible to software executing at EL0, for OS management purposes.

The PE makes no use of this register.

Configuration

This register is banked between TPIDRURO and TPIDRURO_S and TPIDRURO_NS.

AArch32 System register TPIDRURO bits [31:0] are architecturally mapped to AArch64 System register TPIDRRO_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to TPIDRURO are UNDEFINED.

Note

The PE never updates this register.

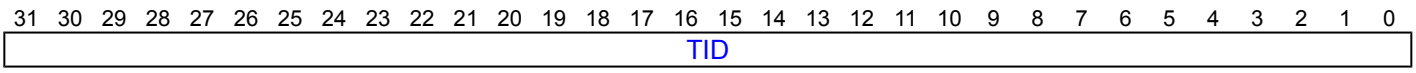
Attributes

TPIDRURO is a 32-bit register.

This register has the following instances:

- TPIDRURO, when EL3 is not implemented or FEAT_AA64 is implemented.
- TPIDRURO_S, when FEAT_AA32EL3 is implemented.
- TPIDRURO_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



TID, bits [31:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDRURO

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        R[t] = TPIDRURO;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TPIDRURO_NS;
    else
        R[t] = TPIDRURO;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TPIDRURO_NS;
    else
        R[t] = TPIDRURO;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = TPIDRURO_S;
    else
        R[t] = TPIDRURO_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TPIDRURO_NS = R[t];
    else
        TPIDRURO = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TPIDRURO_NS = R[t];
    else
        TPIDRURO = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        TPIDRURO_S = R[t];
    else
        TPIDRURO_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDRURW, PL0 Read/Write Software Thread ID Register

The TPIDRURW characteristics are:

Purpose

Provides a location where software executing at EL0 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

This register is banked between TPIDRURW and TPIDRURW_S and TPIDRURW_NS.

AArch32 System register TPIDRURW bits [31:0] are architecturally mapped to AArch64 System register TPIDR_EL0[31:0].

This register is present only when FEAT_AA32 is implemented. Otherwise, direct accesses to TPIDRURW are UNDEFINED.

Note

The PE never updates this register.

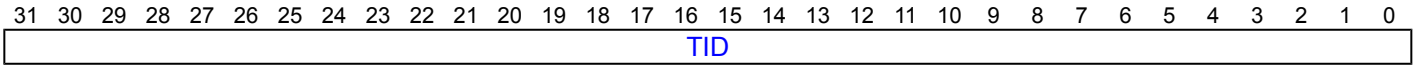
Attributes

TPIDRURW is a 32-bit register.

This register has the following instances:

- TPIDRURW, when EL3 is not implemented or FEAT_AA64 is implemented.
- TPIDRURW_S, when FEAT_AA32EL3 is implemented.
- TPIDRURW_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



TID, bits [31:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDRURW

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGRTR_EL2.TPIDR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        R[t] = TPIDRURW;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TPIDRURW_NS;
    else
        R[t] = TPIDRURW;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TPIDRURW_NS;
    else
        R[t] = TPIDRURW;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = TPIDRURW_S;
    else
        R[t] = TPIDRURW_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2)) &&
!ELIsInHost(EL0) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2)) && HSTR.T13
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_AA64EL1) && !ELUsingAArch32(EL1)) &&
!ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGWTR_EL2.TPIDR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        TPIDRURW = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T13
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T13 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TPIDRURW_NS = R[t];
    else
        TPIDRURW = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TPIDRURW_NS = R[t];
    else
        TPIDRURW = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        TPIDRURW_S = R[t];
    else
        TPIDRURW_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRFCR, Trace Filter Control Register

The TRFCR characteristics are:

Purpose

Provides EL1 controls for Trace.

Configuration

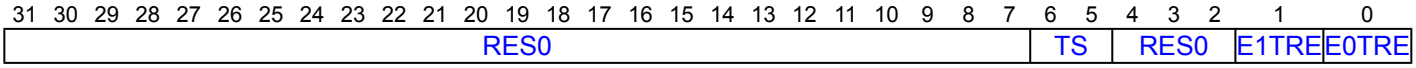
AArch32 System register TRFCR bits [31:0] are architecturally mapped to AArch64 System register TRFCR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented and FEAT_TRF is implemented. Otherwise, direct accesses to TRFCR are UNDEFINED.

Attributes

TRFCR is a 32-bit register.

Field descriptions



Bits [31:7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control. Controls which timebase is used for trace timestamps.

TS	Meaning	Applies when
0b01	Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF.	
0b10	Guest physical timestamp. The traced timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • EL3 is implemented and is using AArch32. • EL3 is implemented, using AArch64, and SCR_EL3.ECVEn == 0b0. • EL2 is using AArch32. • EL2 is using AArch64 and CNTHCTL_EL2.ECV == 0b0. • FEAT_ECV_POFF is not implemented.	When FEAT_ECV is implemented
0b11	Physical timestamp. The traced timestamp is the physical counter value.	

All other values are reserved.

This field is ignored by the PE when any of the following are true:

- EL2 is implemented and HTRFCR.TS != 0b00.
- SelfHostedTraceEnabled() == FALSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:2]

Reserved, RES0.

E1TRE, bit [1]

EL1 Trace Enable.

E1TRE	Meaning
0b0	Tracing is prohibited in PL1 modes.
0b1	Tracing is allowed in PL1 modes.

This field is ignored if SelfHostedTraceEnabled() == FALSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0TRE, bit [0]

EL0 Trace Enable.

E0TRE	Meaning
0b0	Tracing is prohibited at EL0.
0b1	Tracing is allowed at EL0.

This field is ignored if any of the following are true:

- SelfHostedTraceEnabled() == FALSE.
- EL2 is implemented and enabled in the current security state and HCR.TGE == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing TRFCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_TRF)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SDCR.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TTRF == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TTRF
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SDCR.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = TRFCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TTRF ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        R[t] = TRFCR;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TTRF == '1' then
        AArch32.TakeMonitorTrapException();
    else
        R[t] = TRFCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2

0b1111	0b000	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_TRF)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SDCR.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T1 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
MDCR_EL2.TTRF == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HDCR.TTRF
== '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && PSTATE.M !=
M32_Monitor && SDCR.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        TRFCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA64EL3) &&
!ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_AA32EL3) &&
ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) &&
MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && SDCR.TTRF ==
'1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        TRFCR = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TTRF == '1' then
        AArch32.TakeMonitorTrapException();
    else
        TRFCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBCR, Translation Table Base Control Register

The TTBCR characteristics are:

Purpose

The control register for stage 1 of the PL1&0 translation regime. Its controls include:

- Where the VA range is split between addresses translated using TTBR0 and addresses translated using TTBR1.
- The translation table format used by this stage of translation.

From Armv8.2, when the value of TTBCR.{EAE, T2E} is {1, 1}, TTBCR is used with TTBCR2.

Configuration

This register is banked between TTBCR and TTBCR_S and TTBCR_NS.

AArch32 System register TTBCR bits [31:0] are architecturally mapped to AArch64 System register TCR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TTBCR are UNDEFINED.

The current translation table format determines which format of the register is used.

Some RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 then:

- The EAE bit resets to 0 in both the Secure and the Non-secure instances of the register.
- Other reset values apply only to the Secure instance of the register.

Attributes

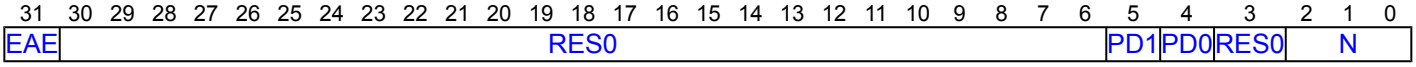
TTBCR is a 32-bit register.

This register has the following instances:

- TTBCR, when EL3 is not implemented or FEAT_AA64 is implemented.
- TTBCR_S, when FEAT_AA32EL3 is implemented.
- TTBCR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When TTBCR.EAE == 0:



EAE, bit [31]

Extended Address Enable.

EAE	Meaning
0b0	Use the VMSAv8-32 translation system with the Short-descriptor translation table format.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [30:6]

Reserved, RES0.

PD1, bit [5]

Translation table walk disable for translations using TTBR1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1.

PD1	Meaning
0b0	Perform translation table walks using TTBR1.
0b1	A TLB miss on an address that is translated using TTBR1 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

PD0, bit [4]

Translation table walk disable for translations using TTBR0. This bit controls whether a translation table walk is performed on a TLB miss for an address that is translated using TTBR0.

PD0	Meaning
0b0	Perform translation table walks using TTBR0.
0b1	A TLB miss on an address that is translated using TTBR0 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [3]

Reserved, RES0.

N, bits [2:0]

Indicate the width of the base address held in TTBR0. In TTBR0, the base address field is bits[31:14-N]. The value of N also determines:

- Whether TTBR0 or TTBR1 is used as the base address for translation table walks.
- The size of the translation table pointed to by TTBR0.

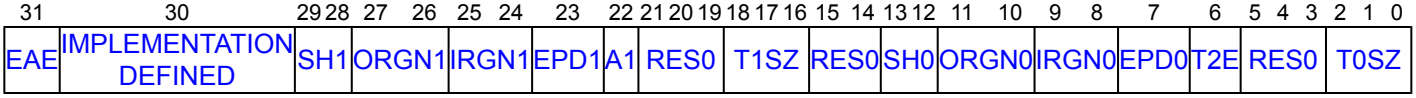
N can take any value from 0 to 7, that is, from 0b000 to 0b111.

When N has its reset value of 0, the translation table base is compatible with Armv5 and Armv6.

The reset behavior of this field is:

- On a Warm reset, this field resets to '000'.

When TTBCR.EAE == 1:



EAE, bit [31]

Extended Address Enable.

EAE	Meaning
0b1	Use the VMSAv8-32 translation system with the Long-descriptor translation table format.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

IMPLEMENTATION DEFINED, bit [30]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using TTBR1.

SH1	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using TTBR1.

ORGN1	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using TTBR1.

IRGN1	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

EPD1, bit [23]

Translation table walk disable for translations using TTBR1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1.

EPD1	Meaning
0b0	Perform translation table walks using TTBR1.
0b1	A TLB miss on an address that is translated using TTBR1 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

A1, bit [22]

Selects whether TTBR0 or TTBR1 defines the ASID.

A1	Meaning
0b0	TTBR0.ASID defines the ASID.
0b1	TTBR1.ASID defines the ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [21:19]

Reserved, RES0.

T1SZ, bits [18:16]

See 'Selecting between TTBR0 and TTBR1, VMSAv8-32 Long-descriptor translation table format' for how TTBCR.{T1SZ, T0SZ} determine the input address ranges and memory region sizes translated using TTBR0 and TTBR1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '000'.

Bits [15:14]

Reserved, RES0.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0.

SH0	Meaning
0b00	Non-shareable
0b10	Outer Shareable
0b11	Inner Shareable

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

EPD0, bit [7]

Translation table walk disable for translations using TTBR0. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR0.

EPD0	Meaning
0b0	Perform translation table walks using TTBR0.
0b1	A TLB miss on an address that is translated using TTBR0 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

T2E, bit [6]

When FEAT_AA32HPD is implemented:

TTBCR2 Enable.

T2E	Meaning
0b0	TTBCR2 is disabled. The contents of TTBCR2 are treated as 0 for all purposes other than reading or writing the register.
0b1	TTBCR2 is enabled.

If TTBCR.EAE==0, then the behavior is as if the bit is 0.

Otherwise:

Reserved, RES0.

Bits [5:3]

Reserved, RES0.

T0SZ, bits [2:0]

See 'Selecting between TTBR0 and TTBR1, VMSAv8-32 Long-descriptor translation table format' for how TTBCR.{T1SZ, T0SZ} determine the input address ranges and memory region sizes translated using TTBR0 and TTBR1.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'000'`.

Accessing TTBCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TTBCR_NS;
    else
        R[t] = TTBCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TTBCR_NS;
    else
        R[t] = TTBCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = TTBCR_S;
    else
        R[t] = TTBCR_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBCR_NS = R[t];
    else
        TTBCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBCR_NS = R[t];
    else
        TTBCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBCR_S = R[t];
        else
            TTBCR_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBCR2, Translation Table Base Control Register 2

The TTBCR2 characteristics are:

Purpose

The second control register for stage 1 of the PL1&0 translation regime.

If FEAT_AA32HPD is not implemented then this register is not implemented and its encoding is UNDEFINED. Otherwise:

- When the value of TTBCR.{EAE, T2E} is not {1, 1} the contents of TTBCR2 are treated as zero for all purposes other than reading or writing the register.
- When the value of TTBCR.{EAE, T2E} is {1, 1} TTBCR2 is used with TTBCR.

Configuration

This register is banked between TTBCR2 and TTBCR2_S and TTBCR2_NS.

AArch32 System register TTBCR2 bits [31:0] are architecturally mapped to AArch64 System register TCR_EL1[63:32].

This register is present only when FEAT_AA32EL1 is implemented and FEAT_AA32HPD is implemented. Otherwise, direct accesses to TTBCR2 are UNDEFINED.

Attributes

TTBCR2 is a 32-bit register.

This register has the following instances:

- TTBCR2, when EL3 is not implemented or FEAT_AA64 is implemented.
- TTBCR2_S, when FEAT_AA32EL3 is implemented.
- TTBCR2_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0													HWU162	HWU161	HWU160	HWU159	HWU062	HWU061	HWU060	HWU059	HPD1	HPD0	RES0								

Bits [31:19]

Reserved, RES0.

HWU162, bit [18]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR1.

HWU162	Meaning
0b0	For translations using TTBR1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [17]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR1.

HWU161	Meaning
0b0	For translations using TTBR1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [16]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR1.

HWU160	Meaning
0b0	For translations using TTBR1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [15]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR1.

HWU159	Meaning
0b0	For translations using TTBR1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [14]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR0.

HWU062	Meaning
0b0	For translations using TTBR0, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [13]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR0.

HWU061	Meaning
0b0	For translations using TTBR0, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [12]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR0.

HWU060	Meaning
0b0	For translations using TTBR0, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [11]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR0.

HWU059	Meaning
0b0	For translations using TTBR0, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [10]

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the translation tables pointed to by TTBR1.

HPD1	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled if TTBCR.T2E == 1.

When disabled, the permissions are treated as if the bits are 0.

The Effective value of this field is 0 if the value of TTBCR.T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HPD0, bit [9]

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the translation tables pointed to by TTBR0.

HPD0	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled if TTBCR.T2E ==1.

When disabled, the permissions are treated is as if the bits are 0.

The Effective value of this field is 0 if the value of TTBCR.T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:0]

Reserved, RES0.

Accessing TTBCR2

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA32HPD)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TTBCR2_NS;
    else
        R[t] = TTBCR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TTBCR2_NS;
    else
        R[t] = TTBCR2;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = TTBCR2_S;
    else
        R[t] = TTBCR2_NS;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA32HPD)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBCR2_NS = R[t];
    else
        TTBCR2 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBCR2_NS = R[t];
    else
        TTBCR2 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBCR2_S = R[t];
        else
            TTBCR2_NS = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR0, Translation Table Base Register 0

The TTBR0 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the PL1&0 translation regime, and other information for this translation regime.

Configuration

This register is banked between TTBR0 and TTBR0_S and TTBR0_NS.

AArch32 System register TTBR0 bits [63:0] are architecturally mapped to AArch64 System register TTBR0_EL1[63:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TTBR0 are UNDEFINED.

TTBR0 is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

TTBCR.EAE determines which TTBR0 format is used:

- TTBCR.EAE == `0b0`: 32-bit format is used. TTBR0[63:32] are ignored.
- TTBCR.EAE == `0b1`: 64-bit format is used.

When EL3 is using AArch32, write access to TTBR0(S) is disabled when the **CP15SDISABLE** signal is asserted HIGH.

Used in conjunction with the TTBCR. When the 64-bit TTBR0 format is used, cacheability and shareability information is held in the TTBCR, not in TTBR0.

Attributes

TTBR0 is a 64-bit register.

This register has the following instances:

- TTBR0, when EL3 is not implemented or FEAT_AA64 is implemented.
- TTBR0_S, when FEAT_AA32EL3 is implemented.
- TTBR0_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When TTBCR.EAE == 0:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
TTB0																									IRGN[0]	NOS	RGN		IMP	S	IRGN[1]
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:32]

Reserved, RES0.

TTB0, bits [31:7]

Translation table base address, bits[31:x], where x is 14-(TTBCR.N). Register bits [x-1:7] are RES0, with the additional requirement that if these bits are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:7] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN, bits [0, 6]

Inner region bits. Bits [0,6] of this register together indicate the Inner Cacheability attributes for the memory associated with the translation table walks. The possible values of IRGN[1:0] are:

IRGN	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Cacheable.
0b11	Normal memory, Inner Write-Back no Write-Allocate Cacheable.

Note

The encoding of the IRGN bits is counter-intuitive, with register bit[6] being IRGN[0] and register bit[0] being IRGN[1]. This encoding is chosen to give a consistent encoding of memory region types and to ensure that software written for ARMv7 without the Multiprocessing Extensions can run unmodified on an implementation that includes the functionality introduced by the ARMv7 Multiprocessing Extensions.

The IRGN field is split as follows:

- IRGN[0] is TTBR0[6].
- IRGN[1] is TTBR0[0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NOS, bit [5]

Not Outer Shareable. When the value of TTBR0.S is 1, indicates whether the memory associated with a translation table walk is Inner Shareable or Outer Shareable:

NOS	Meaning
0b0	Memory is Outer Shareable.
0b1	Memory is Inner Shareable.

This bit is ignored when the value of TTBR0.S is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RGN, bits [4:3]

Region bits. Indicates the Outer cacheability attributes for the memory associated with the translation table walks:

RGN	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Cacheable.
0b11	Normal memory, Outer Write-Back no Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMP, bit [2]

The effect of this bit is IMPLEMENTATION DEFINED. If the translation table implementation does not include any IMPLEMENTATION DEFINED features this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [1]

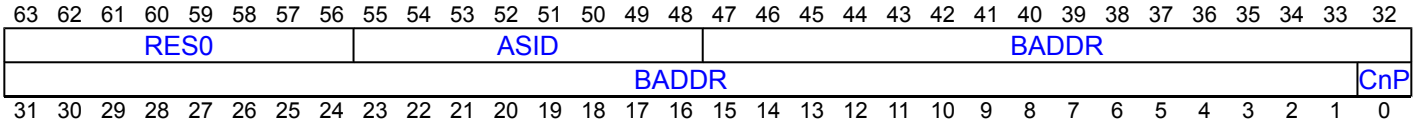
Shareable. Indicates whether the memory associated with the translation table walks is Shareable:

S	Meaning
0b0	Memory is Non-shareable.
0b1	Memory is Shareable. The TTBR0.NOS field indicates whether the memory is Inner Shareable or Outer Shareable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When TTBCR.EAE == 1:



Bits [63:56]

Reserved, RES0.

ASID, bits [55:48]

An ASID for the translation table base address. The TTBCR.A1 field selects either TTBR0.ASID or TTBR1.ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of TTBCR.T0SZ as follows:

- If TTBCR.T0SZ is 0 or 1, x = 5 - TTBCR.T0SZ.
- If TTBCR.T0SZ is greater than 1, x = 14 - TTBCR.T0SZ.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. When TTBCR.EAE ==1, this bit indicates whether each entry that is pointed to by TTBR0 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by this instance of TTBR0, for the current ASID, are permitted to differ from corresponding entries for this instance of TTBR0 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR0.CnP on those other PEs. • The value of TTBCR.EAE on those other PEs. • The value of the current ASID or, for the Non-secure instance of TTBR0, the value of the current VMID.
0b1	The translation table entries pointed to by this instance of TTBR0 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0.CnP is 1 for this instance of TTBR0 and all of the following apply: • The translation table entries are pointed to by this instance of TTBR0. • The value of the applicable TTBCR.EAE field is 1. • The ASID is the same as the current ASID. • For the Non-secure instance of TTBR0, the VMID is the same as the current VMID.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR0.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TTBR0_NS<31:0>;
    else
        R[t] = TTBR0<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TTBR0_NS<31:0>;
    else
        R[t] = TTBR0<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = TTBR0_S<31:0>;
    else
        R[t] = TTBR0_NS<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBR0_NS<31:0> = R[t];
    else
        TTBR0<31:0> = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBR0_NS<31:0> = R[t];
    else
        TTBR0<31:0> = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR0_S<31:0> = R[t];
        else
            TTBR0_NS<31:0> = R[t];
```

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b0010	0b0000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = TTBR0_NS;
    else
        R[t, t2] = TTBR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = TTBR0_NS;
    else
        R[t, t2] = TTBR0;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t, t2] = TTBR0_S;
    else
        R[t, t2] = TTBR0_NS;
```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

coproc	CRm	opc1
0b1111	0b0010	0b0000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBR0_NS = R[t, t2];
    else
        TTBR0 = R[t, t2];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBR0_NS = R[t, t2];
    else
        TTBR0 = R[t, t2];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR0_S = R[t, t2];
        else
            TTBR0_NS = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR1, Translation Table Base Register 1

The TTBR1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the PL1&0 translation regime, and other information for this translation regime.

Configuration

This register is banked between TTBR1 and TTBR1_S and TTBR1_NS.

AArch32 System register TTBR1 bits [63:0] are architecturally mapped to AArch64 System register TTBR1_EL1[63:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to TTBR1 are UNDEFINED.

TTBR1 is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

TTBCR.EAE determines which TTBR1 format is used:

- TTBCR.EAE == `0b0`: 32-bit format is used. TTBR1[63:32] are ignored.
- TTBCR.EAE == `0b1`: 64-bit format is used.

Used in conjunction with the TTBCR. When the 64-bit TTBR1 format is used, cacheability and shareability information is held in the TTBCR, not in TTBR1.

Attributes

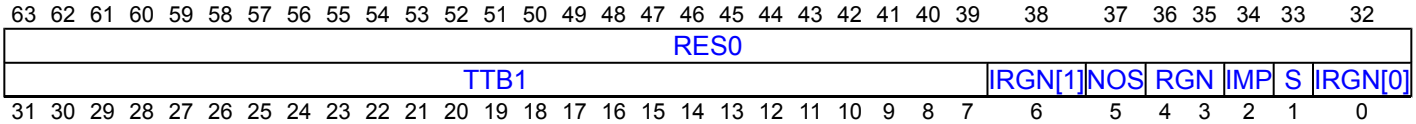
TTBR1 is a 64-bit register.

This register has the following instances:

- TTBR1, when EL3 is not implemented or FEAT_AA64 is implemented.
- TTBR1_S, when FEAT_AA32EL3 is implemented.
- TTBR1_NS, when FEAT_AA32EL3 is implemented.

Field descriptions

When TTBCR.EAE == 0:



Bits [63:32]

Reserved, RES0.

TTB1, bits [31:7]

Translation table base address, bits[31:14]. Register bits [13:7] are RES0, with the additional requirement that if these bits are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [13:7] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN, bits [6, 0]

Inner region bits. IRGN[1:0] indicate the Inner Cacheability attributes for the memory associated with the translation table walks. The possible values of IRGN[1:0] are:

IRGN	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Cacheable.
0b11	Normal memory, Inner Write-Back no Write-Allocate Cacheable.

Note

The encoding of the IRGN bits is counter-intuitive, with register bit[6] being IRGN[0] and register bit[0] being IRGN[1]. This encoding is chosen to give a consistent encoding of memory region types and to ensure that software written for Armv7 without the Multiprocessing Extensions can run unmodified on an implementation that includes the functionality introduced by the ARMv7 Multiprocessing Extensions.

The IRGN field is split as follows:

- IRGN[1] is TTBR1[6].
- IRGN[0] is TTBR1[0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NOS, bit [5]

Not Outer Shareable. When the value of TTBR1.S is 1, indicates whether the memory associated with a translation table walk is Inner Shareable or Outer Shareable:

NOS	Meaning
0b0	Memory is Outer Shareable.
0b1	Memory is Inner Shareable.

This bit is ignored when the value of TTBR1.S is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RGN, bits [4:3]

Region bits. Indicates the Outer cacheability attributes for the memory associated with the translation table walks:

RGN	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Cacheable.
0b11	Normal memory, Outer Write-Back no Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMP, bit [2]

The effect of this bit is IMPLEMENTATION DEFINED. If the translation table implementation does not include any IMPLEMENTATION DEFINED features this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [1]

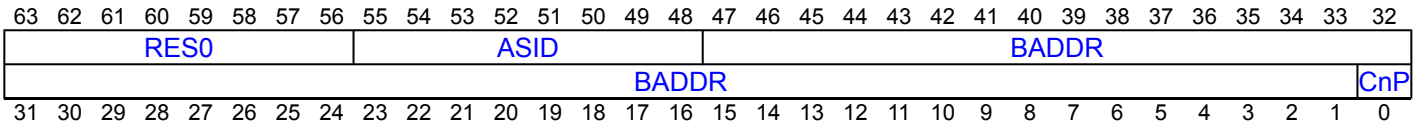
Shareable. Indicates whether the memory associated with the translation table walks is Shareable:

S	Meaning
0b0	Memory is Non-shareable.
0b1	Memory is Shareable. The TTBR1.NOS field indicates whether the memory is Inner Shareable or Outer Shareable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When TTBCR.EAE == 1:



Bits [63:56]

Reserved, RES0.

ASID, bits [55:48]

An ASID for the translation table base address. The TTBCR.A1 field selects either TTBR0.ASID or TTBR1.ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of TTBCR.T1SZ as follows:

- If TTBCR.T1SZ is 0 or 1, x = 5 - TTBCR.T1SZ.
- If TTBCR.T1SZ is greater than 1, x = 14 - TTBCR.T1SZ.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. When TTBCR.EAE ==1, this bit indicates whether each entry that is pointed to by TTBR1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by this instance of TTBR1, for the current ASID, are permitted to differ from corresponding entries for this instance of TTBR1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1.CnP on those other PEs. • The value of TTBCR.EAE on those other PEs. • The value of the current ASID or, for the Non-secure instance of TTBR1, the value of the current VMID.
0b1	The translation table entries pointed to by this instance of TTBR1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1.CnP is 1 for this instance of TTBR1 and all of the following apply: • The translation table entries are pointed to by this instance of TTBR1. • The value of the applicable TTBCR.EAE field is 1. • The ASID is the same as the current ASID. • For the Non-secure instance of TTBR1, the VMID is the same as the current VMID.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TTBR1_NS<31:0>;
    else
        R[t] = TTBR1<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = TTBR1_NS<31:0>;
    else
        R[t] = TTBR1<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = TTBR1_S<31:0>;
    else
        R[t] = TTBR1_NS<31:0>;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS<31:0> = R[t];
    else
        TTBR1<31:0> = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS<31:0> = R[t];
    else
        TTBR1<31:0> = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S<31:0> = R[t];
        else
            TTBR1_NS<31:0> = R[t];
```

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

coproc	CRm	opc1
0b1111	0b0010	0b0001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TRVM ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = TTBR1_NS;
    else
        R[t, t2] = TTBR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t, t2] = TTBR1_NS;
    else
        R[t, t2] = TTBR1;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t, t2] = TTBR1_S;
    else
        R[t, t2] = TTBR1_NS;
```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

coproc	CRm	opc1
0b1111	0b0010	0b0001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.TVM ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS = R[t, t2];
    else
        TTBR1 = R[t, t2];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS = R[t, t2];
    else
        TTBR1 = R[t, t2];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = R[t, t2];
        else
            TTBR1_NS = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VBAR, Vector Base Address Register

The VBAR characteristics are:

Purpose

When high exception vectors are not selected, holds the vector base address for exceptions that are not taken to Monitor mode or to Hyp mode.

Software must program VBAR(NS) with the required initial value as part of the PE boot sequence.

Configuration

This register is banked between VBAR and VBAR_S and VBAR_NS.

AArch32 System register VBAR bits [31:0] are architecturally mapped to AArch64 System register VBAR_EL1[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to VBAR are UNDEFINED.

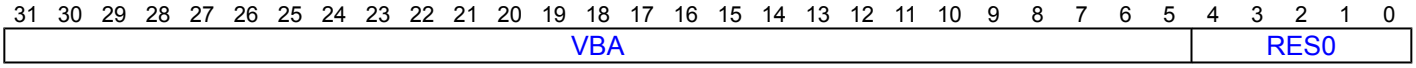
Attributes

VBAR is a 32-bit register.

This register has the following instances:

- VBAR, when EL3 is not implemented or FEAT_AA64 is implemented.
- VBAR_S, when FEAT_AA32EL3 is implemented.
- VBAR_NS, when FEAT_AA32EL3 is implemented.

Field descriptions



VBA, bits [31:5]

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [4:0]

Reserved, RES0.

Accessing VBAR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = VBAR_NS;
    else
        R[t] = VBAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        R[t] = VBAR_NS;
    else
        R[t] = VBAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        R[t] = VBAR_S;
    else
        R[t] = VBAR_NS;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        VBAR_NS = R[t];
    else
        VBAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) then
        VBAR_NS = R[t];
    else
        VBAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            VBAR_S = R[t];
        else
            VBAR_NS = R[t];
```

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VDFSR, Virtual SError Exception Syndrome Register

The VDFSR characteristics are:

Purpose

Provides the syndrome value reported to software on taking a virtual SError exception exception to EL1, or on executing an `ESB` instruction at EL1.

When the virtual SError exception injected using HCR.VA is taken to EL1 using AArch32, then the syndrome value is reported in DFSR.{AET, ExT} and the remainder of DFSR is set as defined by VMSAv8-32. For more information, see The AArch32 Virtual Memory System Architecture.

If the virtual SError exception injected using HCR.VA is deferred by an `ESB` instruction, then the syndrome value is written to VDISR.

Configuration

AArch32 System register VDFSR bits [31:0] are architecturally mapped to AArch64 System register VSESR_EL2[31:0].

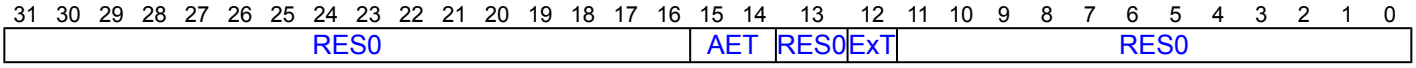
This register is present only when FEAT_RAS is implemented and FEAT_AA32EL1 is implemented. Otherwise, direct accesses to VDFSR are UNDEFINED.

If EL2 is not implemented, then VDFSR is RES0 from Monitor mode when SCR.NS == 1.

Attributes

VDFSR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

AET, bits [15:14]

When a virtual SError exception is taken to EL1 using AArch32, DFSR[15:14] is set to VDFSR.AET.

When a virtual SError exception is deferred by an `ESB` instruction, VDISR[15:14] is set to VDFSR.AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

When a virtual SError exception is taken to EL1 using AArch32, DFSR[12] is set to VDFSR.ExT.

When a virtual SError exception is deferred by an `ESB` instruction, VDISR[12] is set to VDFSR.ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

Accessing VDFSR

Direct reads and writes of VDFSR are UNDEFINED if EL3 is implemented and using AArch32 in all Secure privileged modes other than Monitor mode.

If EL2 is not implemented, then VDFSR is RES0 from Monitor mode when SCR.NS == 1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0101	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = VDFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = VDFSR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0101	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T5
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T5 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VDFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        VDFSR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VDISR, Virtual Deferred Interrupt Status Register

The VDISR characteristics are:

Purpose

Records that an SError exception has been consumed by an ESB instruction.

Configuration

AArch32 System register VDISR bits [31:0] are architecturally mapped to AArch64 System register VDISR_EL2[31:0].

This register is present only when FEAT_AA32EL1 is implemented and FEAT_RAS is implemented. Otherwise, direct accesses to VDISR are UNDEFINED.

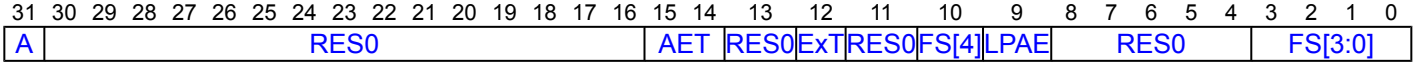
If EL2 is not implemented, then VDISR is RES0 from Monitor mode when SCR.NS == 1.

Attributes

VDISR is a 32-bit register.

Field descriptions

When TTBCR.EAE == 0:



A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from VDFSR.AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from VDFSR.ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault status code. Set to `0b10110` when an `ESB` instruction defers a virtual SError exception.

FS	Meaning
`0b10110`	Asynchronous SError exception.

All other values are reserved.

The FS field is split as follows:

- FS[4] is VDISR[10].
- FS[3:0] is VDISR[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

Format.

Set to TTBCR.EAE when an `ESB` instruction defers a virtual SError exception.

LPAE	Meaning
`0b0`	Using the Short-descriptor translation table format.

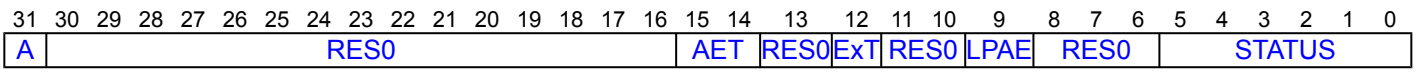
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When TTBCR.EAE == 1:



A, bit [31]

Set to 1 when an `ESB` instruction defers a virtual SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from VDFSR.AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from VDFSR.ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

Format.

Set to TTBCR.EAE when an ESB instruction defers a virtual SError exception.

LPAE	Meaning
0b1	Using the Long-descriptor translation table format.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status code. Set to 0b010001 when an ESB instruction defers a virtual SError exception.

STATUS	Meaning
0b010001	Asynchronous SError exception.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VDISR

Direct reads and writes of VDFSR are UNDEFINED if EL3 is implemented and using AArch32 in all Secure privileged modes other than Monitor mode.

An indirect write to VDISR made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR occurring in program order after the ESB instruction.

If EL2 is not implemented, then VDISR is RES0 from Monitor mode when SCR.NS == 1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_RAS)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = VDISR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = VDISR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b1100	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_RAS)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VDISR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        VDISR = R[t];
```

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
(HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) && IsHCRXEL2Enabled() &&
HCRX_EL2.TMEA == '1')) then
        R[t] = VDISR_EL2<31:0>;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.AMO ==
'1' then
        R[t] = VDISR;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && !Halted()
&& SCR_EL3.EA == '1' then
        R[t] = Zeros(32);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && !Halted() &&
SCR.EA == '1' then
        R[t] = Zeros(32);
    else
        R[t] = DISR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && !Halted() &&
SCR_EL3.EA == '1' then
        R[t] = Zeros(32);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && !Halted() &&
SCR.EA == '1' then
        R[t] = Zeros(32);
    else
        R[t] = DISR;
elsif PSTATE.EL == EL3 then
    R[t] = DISR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b1100	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_RAS) && IsFeatureImplemented(FEAT_AA32EL1)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) &&
(HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) && IsHCRXEL2Enabled() &&
HCRX_EL2.TMEA == '1')) then
        VDISR_EL2 = R[t];
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HCR.AMO ==
'1' then
        VDISR = R[t];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && !Halted()
&& SCR_EL3.EA == '1' then
        return;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && !Halted() &&
SCR.EA == '1' then
        return;
    else
        DISR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64EL3) && !ELUsingAArch32(EL3) && !Halted() &&
SCR_EL3.EA == '1' then
        return;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL3) && ELUsingAArch32(EL3) && !Halted() &&
SCR.EA == '1' then
        return;
    else
        DISR = R[t];
elsif PSTATE.EL == EL3 then
    DISR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VMPIDR, Virtualization Multiprocessor ID Register

The VMPIDR characteristics are:

Purpose

Holds the value of the Virtualization Multiprocessor ID. This is the value returned by Non-secure EL1 reads of MPIDR, which in a multiprocessor system, provides an additional PE identification system.

Configuration

AArch32 System register VMPIDR bits [31:0] are architecturally mapped to AArch64 System register VMPIDR_EL2[31:0].

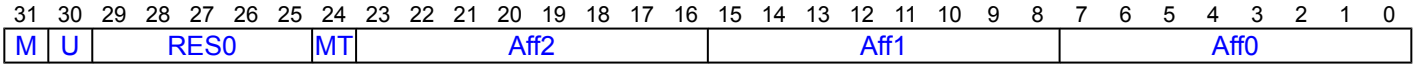
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to VMPIDR are UNDEFINED.

If EL2 is not implemented but EL3 is implemented, this register takes the value of the MPIDR.

Attributes

VMPIDR is a 32-bit register.

Field descriptions



M, bit [31]

Indicates whether this implementation includes the functionality introduced by the Armv7 Multiprocessing Extensions.

M	Meaning
0b0	This implementation does not include the Armv7 Multiprocessing Extensions functionality.
0b1	This implementation includes the Armv7 Multiprocessing Extensions functionality.

Access to this field is **RES1**.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to the value in MPIDR.U.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of Aff0 for more information about affinity levels.

MT	Meaning
0b0	Performance of PEs at the lowest affinity level is largely independent.
0b1	Performance of PEs at the lowest affinity level is very interdependent.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to the value in MPIDR.MT.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to the value in MPIDR.Aff2.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to the value in MPIDR.Aff1.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to the value in MPIDR.Aff0.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing VMPIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = VMPIDR;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        R[t] = MPIDR;
    elsif SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = VMPIDR;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VMPIDR = R[t];
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return;
    elsif SCR.NS == '0' then
        UNDEFINED;
    else
        VMPIDR = R[t];
```

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        R[t] = VMPIDR_EL2<31:0>;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
        R[t] = VMPIDR;
    else
        R[t] = MPIDR;
elsif PSTATE.EL == EL2 then
    R[t] = MPIDR;
elsif PSTATE.EL == EL3 then
    R[t] = MPIDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VPIDR, Virtualization Processor ID Register

The VPIDR characteristics are:

Purpose

Holds the value of the Virtualization Processor ID. This is the value returned by Non-secure EL1 reads of MIDR.

Configuration

AArch32 System register VPIDR bits [31:0] are architecturally mapped to AArch64 System register VPIDR_EL2[31:0].

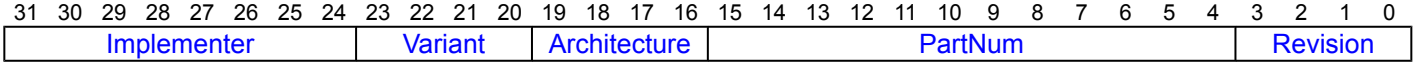
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to VPIDR are UNDEFINED.

If EL2 is not implemented but EL3 is implemented, this register takes the value of the MIDR.

Attributes

VPIDR is a 32-bit register.

Field descriptions



Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm.

Implementer	Meaning
0x00	Reserved for software use.
0x41	Arm Limited.
0x42	Broadcom Corporation.
0x43	Cavium Inc.
0x44	Digital Equipment Corporation.
0x46	Fujitsu Ltd.
0x49	Infineon Technologies AG.
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation.
0x50	Applied Micro Circuits Corporation.
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation.
0xC0	Ampere Computing.

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to the value in MIDR.Implementer.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Variant, bits [23:20]

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to the value in MIDR.Variant.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Architecture, bits [19:16]

Architecture version.

Architecture	Meaning
0b0001	Armv4.
0b0010	Armv4T.
0b0011	Armv5 (obsolete).
0b0100	Armv5T.
0b0101	Armv5TE.
0b0110	Armv5TEJ.
0b0111	Armv6.
0b1111	Architectural features are individually identified in the ID_* registers.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to the value in MIDR.Architecture.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

PartNum, bits [15:4]

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to the value in MIDR.PartNum.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Revision, bits [3:0]

An IMPLEMENTATION DEFINED revision number for the device.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to the value in MIDR.Revision.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing VPIDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = VPIDR;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        R[t] = MIDR;
    elsif SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = VPIDR;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VPIDR = R[t];
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return;
    elsif SCR.NS == '0' then
        UNDEFINED;
    else
        VPIDR = R[t];
```

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc	opc1	CRn	CRm	opc2
0b1111	0b000	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T0
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T0 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        R[t] = VPIDR_EL2<31:0>;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) then
        R[t] = VPIDR;
    else
        R[t] = MIDR;
elsif PSTATE.EL == EL2 then
    R[t] = MIDR;
elsif PSTATE.EL == EL3 then
    R[t] = MIDR;
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VTCR, Virtualization Translation Control Register

The VTCR characteristics are:

Purpose

The control register for stage 2 of the Non-secure PL1&0 translation regime.

Note

This stage of translation always uses the Long-descriptor translation table format.

Configuration

AArch32 System register VTCR bits [31:0] are architecturally mapped to AArch64 System register VTCR_EL2[31:0].

This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to VTCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

VTCR is a 32-bit register.

Field descriptions

31	30 29	28	27	26	25	24 23 22 21 20 19 18 17 16 15 14	13 12	11 10	9 8	7 6	5	4	3 2 1 0
RES1	RES0	HWU62	HWU61	HWU60	HWU59	RES0	SH0	ORGN0	IRGN0	SL0	RES0	S	T0SZ

Bit [31]

Reserved, RES1.

Bits [30:29]

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 2 translation table Block or Page entry.

HWU62	Meaning
0b0	Bit[62] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[62] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 2 translation table Block or Page entry.

HWU61	Meaning
0b0	Bit[61] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[61] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 2 translation table Block or Page entry.

HWU60	Meaning
0b0	Bit[60] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]
When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 2 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[59] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [24:14]

Reserved, RES0.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using VTTBR.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using VTTBR.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using VTTBR.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SL0, bits [7:6]

Starting level for translation table walks using VTTBR.

SL0	Meaning
0b00	Start at level 2
0b01	Start at level 1

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of T0SZ, then a stage 2 level 1 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

S, bit [4]

Sign extension bit. This bit must be programmed to the value of T0SZ[3]. If it is not, then the stage 2 T0SZ value is treated as an UNKNOWN value

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T0SZ, bits [3:0]

The size offset of the memory region addressed by VTTBR. The region size is $2^{(32-T0SZ)}$ bytes.

This field holds a four-bit signed integer value, meaning it supports values from -8 to 7.

Note

This is different from the other translation control registers, where TnSZ holds a three-bit unsigned integer, supporting values from 0 to 7.

If this field is programmed to a value that is not consistent with the programming of SL0 then a stage 2 level 1 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VTCR

Accesses to this register use the following encodings in the System register encoding space:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0010	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t] = VTCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t] = VTCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

coproc	opc1	CRn	CRm	opc2
0b1111	0b100	0b0010	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        VTCR = R[t];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VTTBR, Virtualization Translation Table Base Register

The VTTBR characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 2 of an address translation in the Non-secure PL1&0 translation regime, and other information for this translation regime.

Configuration

AArch32 System register VTTBR bits [63:0] are architecturally mapped to AArch64 System register VTTBR_EL2[63:0].

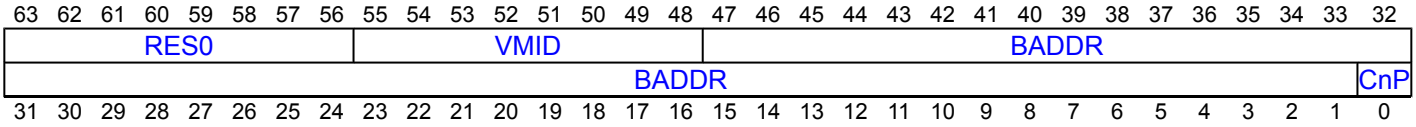
This register is present only when FEAT_AA32EL2 is implemented. Otherwise, direct accesses to VTTBR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

VTTBR is a 64-bit register.

Field descriptions



Bits [63:56]

Reserved, RES0.

VMID, bits [55:48]

The VMID for the translation table.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3 or the highest implemented Exception level is EL2, this field resets to `'00000000'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of VTCR.SL0 and VTCR.T0SZ as follows:

- If VTCR.SL0 is `0b00`, meaning that lookup starts at level 2, then x is 14 - VTCR.T0SZ.
- If VTCR.SL0 is `0b01`, meaning that lookup starts at level 1, then x is 5 - VTCR.T0SZ.
- If VTCR.SL0 is either `0b10` or `0b11` then a stage 2 level 1 Translation fault is generated.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by VTTBR is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VTTBR.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by VTTBR are permitted to differ from the entries for VTTBR for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
0b1	The translation table entries pointed to by VTTBR are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VTTBR.CnP is 1 and the VMID is the same as the current VMID.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the VTTBR.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VTTBRs do not point to the same translation table entries when the VMID value is the same as the current VMID, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing VTTBR

Accesses to this register use the following encodings in the System register encoding space:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

coproc	CRm	opc1
0b1111	0b0010	0b0110

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    R[t, t2] = VTTBR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        R[t, t2] = VTTBR;
```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

coproc	CRm	opc1
0b1111	0b0010	0b0110

```
if !IsFeatureImplemented(FEAT_AA32EL2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T2
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_AA32EL2) && ELUsingAArch32(EL2) && HSTR.T2 ==
'1' then
        AArch32.TakeHypTrapException(0x04);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTTBR = R[t, t2];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        VTTBR = R[t, t2];
```

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

System Register index by instruction and encoding

Below are indexes for registers and operations accessed in the following ways:

For AArch32

- MRC/MCR
- MRRC/MCRR
- MRS/MSR
- VMRS/VMSR

For AArch64

- AT
- BRB
- CFP
- COSP
- CPP
- DC
- DVP
- GCSPOPCX
- GCSPOPX
- GCSPUSHM
- GCSPUSHX
- GCSSS1
- GCSPOPM
- GCSSS2
- IC
- MRRS/MSRR
- MRS/MSR
- TLBI
- TLBIP
- TRCIT

Registers and operations in AArch32

Accessed using MRC/MCR:

coproc	opc1	CRn	CRm	opc2	Access	Mnemonic	Accesses
1110	000	0000	0000	000	RO	DBGDIDR	DBGDIDR
1110	000	0000	0000	010	RW	DBGDTRRXext	DBGDTRRXext
1110	000	0000	0001	000	RO	DBGDSCRint	DBGDSCRint
1110	000	0000	0010	000	RW	DBGDCCINT	DBGDCCINT
1110	000	0000	0010	010	RW	DBGDSCRext	DBGDSCRext
1110	000	0000	0011	010	RW	DBGDTRTXext	DBGDTRTXext
1110	000	0000	0101	000	RO	DBGDTRRXint	DBGDTRRXint
1110	000	0000	0101	000	WO	DBGDTRTXint	DBGDTRTXint
1110	000	0000	0110	000	RW	DBGWFAR	DBGWFAR
1110	000	0000	0110	010	RW	DBGOSECCR	DBGOSECCR
1110	000	0000	0111	000	RW	DBGVCR	DBGVCR
1110	000	0000	m[3:0]	100	RW	DBGBVR<m>	DBGBVR<n>
1110	000	0000	m[3:0]	101	RW	DBGBCR<m>	DBGBCR<n>
1110	000	0000	m[3:0]	110	RW	DBGWVR<m>	DBGWVR<n>
1110	000	0000	m[3:0]	111	RW	DBGWCR<m>	DBGWCR<n>
1110	000	0001	0000	000	RO	DBGDRAR	DBGDRAR
1110	000	0001	0000	100	WO	DBGOSLAR	DBGOSLAR
1110	000	0001	0001	100	RO	DBGOSLSR	DBGOSLSR

coproc	opc1	CRn	CRm	opc2	Access	Mnemonic	Accesses
1110	000	0001	0011	100	RW	DBGOSDLR	DBGOSDLR
1110	000	0001	0100	100	RW	DBGPRCR	DBGPRCR
1110	000	0001	m[3:0]	001	RW	DBGBXVR<m>	DBGBXVR<n>
1110	000	0010	0000	000	RO	DBGDSAR	DBGDSAR
1110	000	0111	0000	111	RO	DBGDEVID2	DBGDEVID2
1110	000	0111	0001	111	RO	DBGDEVID1	DBGDEVID1
1110	000	0111	0010	111	RO	DBGDEVID	DBGDEVID
1110	000	0111	1000	110	RW	DBGCLAIMSET	DBGCLAIMSET
1110	000	0111	1001	110	RW	DBGCLAIMCLR	DBGCLAIMCLR
1110	000	0111	1110	110	RO	DBGAUTHSTATUS	DBGAUTHSTATUS
1110	111	0000	0000	000	RO	JIDR	JIDR
1110	111	0001	0000	000	RW	JOSCR	JOSCR
1110	111	0010	0000	000	RW	JMCR	JMCR
1111	000	0000	0000	000	RO	MIDR	MIDR
1111	000	0000	0000	000	RO	MIDR	VPIDR
1111	000	0000	0000	001	RO	CTR	CTR
1111	000	0000	0000	010	RO	TCMTR	TCMTR
1111	000	0000	0000	011	RO	TLBTR	TLBTR
1111	000	0000	0000	101	RO	MPIDR	MPIDR
1111	000	0000	0000	101	RO	MPIDR	VMPIDR
1111	000	0000	0000	110	RO	REVIDR	REVIDR
1111	000	0000	0001	000	RO	ID_PFR0	ID_PFR0
1111	000	0000	0001	001	RO	ID_PFR1	ID_PFR1
1111	000	0000	0001	010	RO	ID_DFR0	ID_DFR0
1111	000	0000	0001	011	RO	ID_AFR0	ID_AFR0
1111	000	0000	0001	100	RO	ID_MMFR0	ID_MMFR0
1111	000	0000	0001	101	RO	ID_MMFR1	ID_MMFR1
1111	000	0000	0001	110	RO	ID_MMFR2	ID_MMFR2
1111	000	0000	0001	111	RO	ID_MMFR3	ID_MMFR3
1111	000	0000	0010	000	RO	ID_ISAR0	ID_ISAR0
1111	000	0000	0010	001	RO	ID_ISAR1	ID_ISAR1
1111	000	0000	0010	010	RO	ID_ISAR2	ID_ISAR2
1111	000	0000	0010	011	RO	ID_ISAR3	ID_ISAR3
1111	000	0000	0010	100	RO	ID_ISAR4	ID_ISAR4
1111	000	0000	0010	101	RO	ID_ISAR5	ID_ISAR5
1111	000	0000	0010	110	RO	ID_MMFR4	ID_MMFR4
1111	000	0000	0010	111	RO	ID_ISAR6	ID_ISAR6
1111	000	0000	0011	100	RO	ID_PFR2	ID_PFR2
1111	000	0000	0011	101	RO	ID_DFR1	ID_DFR1
1111	000	0000	0011	110	RO	ID_MMFR5	ID_MMFR5
1111	000	0001	0000	000	RW	SCTLR	SCTLR
1111	000	0001	0000	001	RW	ACTLR	ACTLR
1111	000	0001	0000	010	RW	CPACR	CPACR
1111	000	0001	0000	011	RW	ACTLR2	ACTLR2
1111	000	0001	0001	000	RW	SCR	SCR
1111	000	0001	0001	001	RW	SDER	SDER
1111	000	0001	0001	010	RW	NSACR	NSACR
1111	000	0001	0010	001	RW	TRFCR	TRFCR
1111	000	0001	0011	001	RW	SDCR	SDCR

coproc	opc1	CRn	CRm	opc2	Access	Mnemonic	Accesses
1111	000	0010	0000	000	RW	TTBR0	TTBR0
1111	000	0010	0000	001	RW	TTBR1	TTBR1
1111	000	0010	0000	010	RW	TTBCR	TTBCR
1111	000	0010	0000	011	RW	TTBCR2	TTBCR2
1111	000	0011	0000	000	RW	DACR	DACR
1111	000	0100	0110	000	RW	ICC_PMR	ICC_PMR
1111	000	0100	0110	000	RW	ICC_PMR	ICV_PMR
1111	000	0101	0000	000	RW	DFSR	DFSR
1111	000	0101	0000	001	RW	IFSR	IFSR
1111	000	0101	0001	000	RW	ADFSR	ADFSR
1111	000	0101	0001	001	RW	AIFSR	AIFSR
1111	000	0101	0011	000	RO	ERRIDR	ERRIDR
1111	000	0101	0011	001	RW	ERRSELR	ERRSELR
1111	000	0101	0100	000	RO	ERXFR	ERXFR
1111	000	0101	0100	001	RW	ERXCTLR	ERXCTLR
1111	000	0101	0100	010	RW	ERXSTATUS	ERXSTATUS
1111	000	0101	0100	011	RW	ERXADDR	ERXADDR
1111	000	0101	0100	100	RO	ERXFR2	ERXFR2
1111	000	0101	0100	101	RW	ERXCTLR2	ERXCTLR2
1111	000	0101	0100	111	RW	ERXADDR2	ERXADDR2
1111	000	0101	0101	000	RW	ERXMISC0	ERXMISC0
1111	000	0101	0101	001	RW	ERXMISC1	ERXMISC1
1111	000	0101	0101	010	RW	ERXMISC4	ERXMISC4
1111	000	0101	0101	011	RW	ERXMISC5	ERXMISC5
1111	000	0101	0101	100	RW	ERXMISC2	ERXMISC2
1111	000	0101	0101	101	RW	ERXMISC3	ERXMISC3
1111	000	0101	0101	110	RW	ERXMISC6	ERXMISC6
1111	000	0101	0101	111	RW	ERXMISC7	ERXMISC7
1111	000	0110	0000	000	RW	DFAR	DFAR
1111	000	0110	0000	010	RW	IFAR	IFAR
1111	000	0111	0001	000	WO	ICIALLUIS	ICIALLUIS
1111	000	0111	0001	110	WO	BPIALLIS	BPIALLIS
1111	000	0111	0011	100	WO	CFPRCTX	CFPRCTX
1111	000	0111	0011	101	WO	DVPRCTX	DVPRCTX
1111	000	0111	0011	110	WO	COSPRCTX	COSPRCTX
1111	000	0111	0011	111	WO	CPPRCTX	CPPRCTX
1111	000	0111	0100	000	RW	PAR	PAR
1111	000	0111	0101	000	WO	ICIALLU	ICIALLU
1111	000	0111	0101	001	WO	ICIMVAU	ICIMVAU
1111	000	0111	0101	100	WO	CP15ISB	CP15ISB
1111	000	0111	0101	110	WO	BPIALL	BPIALL
1111	000	0111	0101	111	WO	BPIMVA	BPIMVA
1111	000	0111	0110	001	WO	DCIMVAC	DCIMVAC
1111	000	0111	0110	010	WO	DCISW	DCISW
1111	000	0111	1000	000	WO	ATS1CPR	ATS1CPR
1111	000	0111	1000	001	WO	ATS1CPW	ATS1CPW
1111	000	0111	1000	010	WO	ATS1CUR	ATS1CUR
1111	000	0111	1000	011	WO	ATS1CUW	ATS1CUW
1111	000	0111	1000	100	WO	ATS12NSOPR	ATS12NSOPR

coproc	opc1	CRn	CRm	opc2	Access	Mnemonic	Accesses
1111	000	0111	1000	101	WO	ATS12NSOPW	ATS12NSOPW
1111	000	0111	1000	110	WO	ATS12NSOUR	ATS12NSOUR
1111	000	0111	1000	111	WO	ATS12NSOUW	ATS12NSOUW
1111	000	0111	1001	000	WO	ATS1CPRP	ATS1CPRP
1111	000	0111	1001	001	WO	ATS1CPWP	ATS1CPWP
1111	000	0111	1010	001	WO	DCCMVAC	DCCMVAC
1111	000	0111	1010	010	WO	DCCSW	DCCSW
1111	000	0111	1010	100	WO	CP15DSB	CP15DSB
1111	000	0111	1010	101	WO	CP15DMB	CP15DMB
1111	000	0111	1011	001	WO	DCCMVAU	DCCMVAU
1111	000	0111	1110	001	WO	DCCIMVAC	DCCIMVAC
1111	000	0111	1110	010	WO	DCCISW	DCCISW
1111	000	1000	0011	000	WO	TLBIALLIS	TLBIALLIS
1111	000	1000	0011	001	WO	TLBIMVAIS	TLBIMVAIS
1111	000	1000	0011	010	WO	TLBIASIDIS	TLBIASIDIS
1111	000	1000	0011	011	WO	TLBIMVAAIS	TLBIMVAAIS
1111	000	1000	0011	101	WO	TLBIMVALIS	TLBIMVALIS
1111	000	1000	0011	111	WO	TLBIMVAALIS	TLBIMVAALIS
1111	000	1000	0101	000	WO	ITLBIALL	ITLBIALL
1111	000	1000	0101	001	WO	ITLBIMVA	ITLBIMVA
1111	000	1000	0101	010	WO	ITLBIASID	ITLBIASID
1111	000	1000	0110	000	WO	DTLBIALL	DTLBIALL
1111	000	1000	0110	001	WO	DTLBIMVA	DTLBIMVA
1111	000	1000	0110	010	WO	DTLBIASID	DTLBIASID
1111	000	1000	0111	000	WO	TLBIALL	TLBIALL
1111	000	1000	0111	001	WO	TLBIMVA	TLBIMVA
1111	000	1000	0111	010	WO	TLBIASID	TLBIASID
1111	000	1000	0111	011	WO	TLBIMVAA	TLBIMVAA
1111	000	1000	0111	101	WO	TLBIMVAL	TLBIMVAL
1111	000	1000	0111	111	WO	TLBIMVAAL	TLBIMVAAL
1111	000	1001	1100	000	RW	PMCR	PMCR
1111	000	1001	1100	001	RW	PMCNTENSET	PMCNTENSET
1111	000	1001	1100	010	RW	PMCNTENCLR	PMCNTENCLR
1111	000	1001	1100	011	RW	PMOVSR	PMOVSR
1111	000	1001	1100	100	WO	PMSWINC	PMSWINC
1111	000	1001	1100	101	RW	PMSELR	PMSELR
1111	000	1001	1100	110	RO	PMCEID0	PMCEID0
1111	000	1001	1100	111	RO	PMCEID1	PMCEID1
1111	000	1001	1101	000	RW	PMCCNTR	PMCCNTR
1111	000	1001	1101	001	RW	PMXEVTYPER	PMXEVTYPER
1111	000	1001	1101	010	RW	PMXEVCNTR	PMXEVCNTR
1111	000	1001	1110	000	RW	PMUSERENR	PMUSERENR
1111	000	1001	1110	001	RW	PMINTENSET	PMINTENSET
1111	000	1001	1110	010	RW	PMINTENCLR	PMINTENCLR
1111	000	1001	1110	011	RW	PMOVSSET	PMOVSSET
1111	000	1001	1110	100	RO	PMCEID2	PMCEID2
1111	000	1001	1110	101	RO	PMCEID3	PMCEID3
1111	000	1001	1110	110	RO	PMMIR	PMMIR
1111	000	1010	0010	000	RW	PRRR-MAIR0	MAIR0

coproc	opc1	CRn	CRm	opc2	Access	Mnemonic	Accesses
1111	000	1010	0010	000	RW	PRRR-MAIR0	PRRR
1111	000	1010	0010	001	RW	NMRR-MAIR1	MAIR1
1111	000	1010	0010	001	RW	NMRR-MAIR1	NMRR
1111	000	1010	0011	000	RW	AMAIR0	AMAIR0
1111	000	1010	0011	001	RW	AMAIR1	AMAIR1
1111	000	1100	0000	000	RW	VBAR	VBAR
1111	000	1100	0000	001	RO	RVBAR-MVBAR	RVBAR
1111	000	1100	0000	001	RW	RVBAR-MVBAR	MVBAR
1111	000	1100	0000	010	RW	RMR	RMR
1111	000	1100	0001	000	RO	ISR	ISR
1111	000	1100	0001	001	RW	DISR	DISR
1111	000	1100	0001	001	RW	DISR	VDISR
1111	000	1100	1000	000	RO	ICC_IAR0	ICC_IAR0
1111	000	1100	1000	000	RO	ICC_IAR0	ICV_IAR0
1111	000	1100	1000	001	WO	ICC_EOIR0	ICC_EOIR0
1111	000	1100	1000	001	WO	ICC_EOIR0	ICV_EOIR0
1111	000	1100	1000	010	RO	ICC_HPPIR0	ICC_HPPIR0
1111	000	1100	1000	010	RO	ICC_HPPIR0	ICV_HPPIR0
1111	000	1100	1000	011	RW	ICC_BPR0	ICC_BPR0
1111	000	1100	1000	011	RW	ICC_BPR0	ICV_BPR0
1111	000	1100	1000	1:m[1:0]	RW	ICC_AP0R<m>	ICC_AP0R<n>
1111	000	1100	1000	1:m[1:0]	RW	ICC_AP0R<m>	ICV_AP0R<n>
1111	000	1100	1001	0:m[1:0]	RW	ICC_AP1R<m>	ICC_AP1R<n>
1111	000	1100	1001	0:m[1:0]	RW	ICC_AP1R<m>	ICV_AP1R<n>
1111	000	1100	1011	001	WO	ICC_DIR	ICC_DIR
1111	000	1100	1011	001	WO	ICC_DIR	ICV_DIR
1111	000	1100	1011	011	RO	ICC_RPR	ICC_RPR
1111	000	1100	1011	011	RO	ICC_RPR	ICV_RPR
1111	000	1100	1100	000	RO	ICC_IAR1	ICC_IAR1
1111	000	1100	1100	000	RO	ICC_IAR1	ICV_IAR1
1111	000	1100	1100	001	WO	ICC_EOIR1	ICC_EOIR1
1111	000	1100	1100	001	WO	ICC_EOIR1	ICV_EOIR1
1111	000	1100	1100	010	RO	ICC_HPPIR1	ICC_HPPIR1
1111	000	1100	1100	010	RO	ICC_HPPIR1	ICV_HPPIR1
1111	000	1100	1100	011	RW	ICC_BPR1	ICC_BPR1
1111	000	1100	1100	011	RW	ICC_BPR1	ICV_BPR1
1111	000	1100	1100	100	RW	ICC_CTLR	ICC_CTLR
1111	000	1100	1100	100	RW	ICC_CTLR	ICV_CTLR
1111	000	1100	1100	101	RW	ICC_SRE	ICC_SRE
1111	000	1100	1100	110	RW	ICC_IGRPEN0	ICC_IGRPEN0
1111	000	1100	1100	110	RW	ICC_IGRPEN0	ICV_IGRPEN0
1111	000	1100	1100	111	RW	ICC_IGRPEN1	ICC_IGRPEN1
1111	000	1100	1100	111	RW	ICC_IGRPEN1	ICV_IGRPEN1
1111	000	1101	0000	000	RW	FCSEIDR	FCSEIDR
1111	000	1101	0000	001	RW	CONTEXTIDR	CONTEXTIDR
1111	000	1101	0000	010	RW	TPIDRURW	TPIDRURW
1111	000	1101	0000	011	RW	TPIDRURO	TPIDRURO
1111	000	1101	0000	100	RW	TPIDRPRW	TPIDRPRW
1111	000	1101	0010	000	RW	AMCR	AMCR

coproc	opc1	CRn	CRm	opc2	Access	Mnemonic	Accesses
1111	000	1101	0010	001	RO	AMCFGR	AMCFGR
1111	000	1101	0010	010	RO	AMCGCR	AMCGCR
1111	000	1101	0010	011	RW	AMUSERENR	AMUSERENR
1111	000	1101	0010	100	RW	AMCNTENCLR0	AMCNTENCLR0
1111	000	1101	0010	101	RW	AMCNTENSET0	AMCNTENSET0
1111	000	1101	0011	000	RW	AMCNTENCLR1	AMCNTENCLR1
1111	000	1101	0011	001	RW	AMCNTENSET1	AMCNTENSET1
1111	000	1101	011:m[3]	m[2:0]	RO	AMEVTYPER0<m>	AMEVTYPER0<n>
1111	000	1101	111:m[3]	m[2:0]	RW	AMEVTYPER1<m>	AMEVTYPER1<n>
1111	000	1110	0000	000	RW	CNTFRQ	CNTFRQ
1111	000	1110	0001	000	RW	CNTKCTL	CNTKCTL
1111	000	1110	0010	000	RW	CNTP_TVAL	CNTHP_TVAL
1111	000	1110	0010	000	RW	CNTP_TVAL	CNTHPS_TVAL
1111	000	1110	0010	000	RW	CNTP_TVAL	CNTP_TVAL
1111	000	1110	0010	001	RW	CNTP_CTL	CNTHP_CTL
1111	000	1110	0010	001	RW	CNTP_CTL	CNTHPS_CTL
1111	000	1110	0010	001	RW	CNTP_CTL	CNTP_CTL
1111	000	1110	0011	000	RW	CNTV_TVAL	CNTHV_TVAL
1111	000	1110	0011	000	RW	CNTV_TVAL	CNTHVS_TVAL
1111	000	1110	0011	000	RW	CNTV_TVAL	CNTV_TVAL
1111	000	1110	0011	001	RW	CNTV_CTL	CNTHV_CTL
1111	000	1110	0011	001	RW	CNTV_CTL	CNTHVS_CTL
1111	000	1110	0011	001	RW	CNTV_CTL	CNTV_CTL
1111	000	1110	10:m[4:3]	m[2:0]	RW	PMEVCNTR<m>	PMEVCNTR<n>
1111	000	1110	1111	111	RW	PMCCFILTR	PMCCFILTR
1111	000	1110	11:m[4:3]	m[2:0]	RW	PMEVTYPER<m>	PMEVTYPER<n>
1111	001	0000	0000	000	RO	CCSIDR	CCSIDR
1111	001	0000	0000	001	RO	CLIDR	CLIDR
1111	001	0000	0000	010	RO	CCSIDR2	CCSIDR2
1111	001	0000	0000	111	RO	AIDR	AIDR
1111	010	0000	0000	000	RW	CSSELR	CSSELR
1111	011	0100	0101	000	RW	DSPSR	DSPSR
1111	011	0100	0101	001	RW	DLR	DLR
1111	011	0100	0101	010	RW	DSPSR2	DSPSR2
1111	100	0000	0000	000	RW	VPIDR	VPIDR
1111	100	0000	0000	101	RW	VMPIDR	VMPIDR
1111	100	0001	0000	000	RW	HSCTLR	HSCTLR
1111	100	0001	0000	001	RW	HACTLR	HACTLR
1111	100	0001	0000	011	RW	HACTLR2	HACTLR2
1111	100	0001	0001	000	RW	HCR	HCR
1111	100	0001	0001	001	RW	HDCR	HDCR
1111	100	0001	0001	010	RW	HCPTR	HCPTR
1111	100	0001	0001	011	RW	HSTR	HSTR
1111	100	0001	0001	100	RW	HCR2	HCR2
1111	100	0001	0001	111	RW	HACR	HACR
1111	100	0001	0010	001	RW	HTRFCR	HTRFCR
1111	100	0010	0000	010	RW	HTCR	HTCR
1111	100	0010	0001	010	RW	VTCR	VTCR
1111	100	0101	0001	000	RW	HADFSR	HADFSR

coproc	opc1	CRn	CRm	opc2	Access	Mnemonic	Accesses
1111	100	0101	0001	001	RW	HAIFSR	HAIFSR
1111	100	0101	0010	000	RW	HSR	HSR
1111	100	0101	0010	011	RW	VDFSR	VDFSR
1111	100	0110	0000	000	RW	HDFAR	HDFAR
1111	100	0110	0000	010	RW	HIFAR	HIFAR
1111	100	0110	0000	100	RW	HPFAR	HPFAR
1111	100	0111	1000	000	WO	ATS1HR	ATS1HR
1111	100	0111	1000	001	WO	ATS1HW	ATS1HW
1111	100	1000	0000	001	WO	TLBIIPAS2IS	TLBIIPAS2IS
1111	100	1000	0000	101	WO	TLBIIPAS2LIS	TLBIIPAS2LIS
1111	100	1000	0011	000	WO	TLBIALLHIS	TLBIALLHIS
1111	100	1000	0011	001	WO	TLBIMVAHIS	TLBIMVAHIS
1111	100	1000	0011	100	WO	TLBIALLNSNHIS	TLBIALLNSNHIS
1111	100	1000	0011	101	WO	TLBIMVALHIS	TLBIMVALHIS
1111	100	1000	0100	001	WO	TLBIIPAS2	TLBIIPAS2
1111	100	1000	0100	101	WO	TLBIIPAS2L	TLBIIPAS2L
1111	100	1000	0111	000	WO	TLBIALLH	TLBIALLH
1111	100	1000	0111	001	WO	TLBIMVAH	TLBIMVAH
1111	100	1000	0111	100	WO	TLBIALLNSNH	TLBIALLNSNH
1111	100	1000	0111	101	WO	TLBIMVALH	TLBIMVALH
1111	100	1010	0010	000	RW	HMAIR0	HMAIR0
1111	100	1010	0010	001	RW	HMAIR1	HMAIR1
1111	100	1010	0011	000	RW	HAMAIR0	HAMAIR0
1111	100	1010	0011	001	RW	HAMAIR1	HAMAIR1
1111	100	1100	0000	000	RW	HVBAR	HVBAR
1111	100	1100	0000	010	RW	HRMR	HRMR
1111	100	1100	0001	001	RW	VDISR	VDISR
1111	100	1100	1000	0:m[1:0]	RW	ICH_AP0R<m>	ICH_AP0R<n>
1111	100	1100	1001	0:m[1:0]	RW	ICH_AP1R<m>	ICH_AP1R<n>
1111	100	1100	1001	101	RW	ICC_HSRE	ICC_HSRE
1111	100	1100	1011	000	RW	ICH_HCR	ICH_HCR
1111	100	1100	1011	001	RO	ICH_VTR	ICH_VTR
1111	100	1100	1011	010	RO	ICH_MISR	ICH_MISR
1111	100	1100	1011	011	RO	ICH_EISR	ICH_EISR
1111	100	1100	1011	101	RO	ICH_ELRSR	ICH_ELRSR
1111	100	1100	1011	111	RW	ICH_VMCR	ICH_VMCR
1111	100	1100	110:m[3]	m[2:0]	RW	ICH_LR<m>	ICH_LR<n>
1111	100	1100	111:m[3]	m[2:0]	RW	ICH_LRC<m>	ICH_LRC<n>
1111	100	1101	0000	010	RW	HTPIDR	HTPIDR
1111	100	1110	0001	000	RW	CNTHCTL	CNTHCTL
1111	100	1110	0010	000	RW	CNTHP_TVAL	CNTHP_TVAL
1111	100	1110	0010	001	RW	CNTHP_CTL	CNTHP_CTL
1111	110	1100	1100	100	RW	ICC_MCTLR	ICC_MCTLR
1111	110	1100	1100	101	RW	ICC_MSRE	ICC_MSRE
1111	110	1100	1100	111	RW	ICC_MGRPEN1	ICC_MGRPEN1

Accessed using MRRC/MCRR:

coproc	opc1	CRm	Access	Mnemonic	Accesses
1110	0000	0001	RO	DBGDRAR	DBGDRAR
1110	0000	0010	RO	DBGDSAR	DBGDSAR
1111	0000	0010	RW	TTBR0	TTBR0
1111	0000	0111	RW	PAR	PAR
1111	0000	1001	RW	PMCCNTR	PMCCNTR
1111	0000	1100	WO	ICC_SGI1R	ICC_SGI1R
1111	0000	1110	RO	CNTPCT	CNTPCT
1111	0001	0010	RW	TTBR1	TTBR1
1111	0001	1100	WO	ICC_ASGI1R	ICC_ASGI1R
1111	0001	1110	RO	CNTVCT	CNTVCT
1111	0010	1100	WO	ICC_SGI0R	ICC_SGI0R
1111	0010	1110	RW	CNTP_CVAL	CNTHP_CVAL
1111	0010	1110	RW	CNTP_CVAL	CNTHPS_CVAL
1111	0010	1110	RW	CNTP_CVAL	CNTP_CVAL
1111	0011	1110	RW	CNTV_CVAL	CNTHV_CVAL
1111	0011	1110	RW	CNTV_CVAL	CNTHVS_CVAL
1111	0011	1110	RW	CNTV_CVAL	CNTV_CVAL
1111	0100	0010	RW	HTTBR	HTTBR
1111	0100	1110	RW	CNTVOFF	CNTVOFF
1111	0110	0010	RW	VTTBR	VTTBR
1111	0110	1110	RW	CNTHP_CVAL	CNTHP_CVAL
1111	0:m[2:0]	000:m[3]	RW	AMEVCNTR0<m>	AMEVCNTR0<n>
1111	0:m[2:0]	010:m[3]	RW	AMEVCNTR1<m>	AMEVCNTR1<n>
1111	1000	1110	RO	CNTPCTSS	CNTPCTSS
1111	1001	1110	RO	CNTVCTSS	CNTVCTSS

Accessed using MRS/MSR:

R	M	M1	Mnemonic
0	1	1110	ELR_hyp
1	0	1110	SPSR_fiq
1	1	0000	SPSR_irq
1	1	0010	SPSR_svc
1	1	0100	SPSR_abt
1	1	0110	SPSR_und
1	1	1100	SPSR_mon
1	1	1110	SPSR_hyp

Accessed using VMRS/VMSR:

reg	Access	Mnemonic	Accesses
0000	RW	FPSID	FPSID
0001	RW	FPSCR	FPSCR
0101	RO	MVFR2	MVFR2
0110	RO	MVFR1	MVFR1
0111	RO	MVFR0	MVFR0
1000	RW	FPEXC	FPEXC

Registers and operations in AArch64

Accessed using AT:

op0	op1	CRn	CRm	op2	Mnemonic
01	000	0111	1000	000	AT S1E1R
01	000	0111	1000	001	AT S1E1W
01	000	0111	1000	010	AT S1E0R
01	000	0111	1000	011	AT S1E0W
01	000	0111	1001	000	AT S1E1RP
01	000	0111	1001	001	AT S1E1WP
01	000	0111	1001	010	AT S1E1A
01	100	0111	1000	000	AT S1E2R
01	100	0111	1000	001	AT S1E2W
01	100	0111	1000	100	AT S12E1R
01	100	0111	1000	101	AT S12E1W
01	100	0111	1000	110	AT S12E0R
01	100	0111	1000	111	AT S12E0W
01	100	0111	1001	010	AT S1E2A
01	110	0111	1000	000	AT S1E3R
01	110	0111	1000	001	AT S1E3W
01	110	0111	1001	010	AT S1E3A

Accessed using BRB:

op0	op1	CRn	CRm	op2	Mnemonic
01	001	0111	0010	100	BRB IALL
01	001	0111	0010	101	BRB INJ

Accessed using CFP:

op0	op1	CRn	CRm	op2	Mnemonic
01	011	0111	0011	100	CFP RCTX

Accessed using COSP:

op0	op1	CRn	CRm	op2	Mnemonic
01	011	0111	0011	110	COSP RCTX

Accessed using CPP:

op0	op1	CRn	CRm	op2	Mnemonic
01	011	0111	0011	111	CPP RCTX

Accessed using DC:

op0	op1	CRn	CRm	op2	Mnemonic
01	000	0111	0110	001	DC IVAC
01	000	0111	0110	010	DC ISW
01	000	0111	0110	011	DC IGVAC

op0	op1	CRn	CRm	op2	Mnemonic
01	000	0111	0110	100	DC IGSW
01	000	0111	0110	101	DC IGDVAC
01	000	0111	0110	110	DC IGDSW
01	000	0111	1010	010	DC CSW
01	000	0111	1010	100	DC CGSW
01	000	0111	1010	110	DC CGDSW
01	000	0111	1110	010	DC CISW
01	000	0111	1110	100	DC CIGSW
01	000	0111	1110	110	DC CIGDSW
01	000	0111	1111	001	DC CIVAPS
01	000	0111	1111	101	DC CIGDVAPS
01	011	0111	0100	001	DC ZVA
01	011	0111	0100	011	DC GVA
01	011	0111	0100	100	DC GZVA
01	011	0111	1010	001	DC CVAC
01	011	0111	1010	011	DC CGVAC
01	011	0111	1010	101	DC CGDVAC
01	011	0111	1011	000	DC CVAOC
01	011	0111	1011	001	DC CVAU
01	011	0111	1011	111	DC CGDVAOC
01	011	0111	1100	001	DC CVAP
01	011	0111	1100	011	DC CGVAP
01	011	0111	1100	101	DC CGDVAP
01	011	0111	1101	001	DC CVADP
01	011	0111	1101	011	DC CGVADP
01	011	0111	1101	101	DC CGDVADP
01	011	0111	1110	001	DC CIVAC
01	011	0111	1110	011	DC CIGVAC
01	011	0111	1110	101	DC CIGDVAC
01	011	0111	1111	000	DC CIVAOC
01	011	0111	1111	111	DC CIGDVAOC
01	100	0111	1110	000	DC CIPAE
01	100	0111	1110	111	DC CIGDPAE
01	110	0111	1110	001	DC CIPAPA
01	110	0111	1110	101	DC CIGDPAPA

Accessed using DVP:

op0	op1	CRn	CRm	op2	Mnemonic
01	011	0111	0011	101	DVP RCTX

Accessed using GCSPOPCX:

op0	op1	CRn	CRm	op2	Mnemonic
01	000	0111	0111	101	GCSPOPCX

Accessed using GCSPOPX:

op0	op1	CRn	CRm	op2	Mnemonic
01	000	0111	0111	110	GCSPOPX

Accessed using GCSPUSHM:

op0	op1	CRn	CRm	op2	Mnemonic
01	011	0111	0111	000	GCSPUSHM

Accessed using GCSPUSHX:

op0	op1	CRn	CRm	op2	Mnemonic
01	000	0111	0111	100	GCSPUSHX

Accessed using GCSSS1:

op0	op1	CRn	CRm	op2	Mnemonic
01	011	0111	0111	010	GCSSS1

Accessed using GCSPOPM:

op0	op1	CRn	CRm	op2	Mnemonic
01	011	0111	0111	001	GCSPOPM

Accessed using GCSSS2:

op0	op1	CRn	CRm	op2	Mnemonic
01	011	0111	0111	011	GCSSS2

Accessed using IC:

op0	op1	CRn	CRm	op2	Mnemonic
01	000	0111	0001	000	IC IALLUIS
01	000	0111	0101	000	IC IALLU
01	011	0111	0101	001	IC IVAU

Accessed using MRRS/MSRR:

op0	op1	CRn	CRm	op2	Mnemonic
11	000	0010	0000	000	TTBR0_EL1
11	000	0010	0000	001	TTBR1_EL1
11	000	0111	0100	000	PAR_EL1
11	000	1101	0000	011	RCWSMASK_EL1
11	000	1101	0000	110	RCWMASK_EL1
11	100	0010	0000	000	TTBR0_EL2
11	100	0010	0000	001	TTBR1_EL2
11	100	0010	0001	000	VTTBR_EL2
11	101	0010	0000	000	TTBR0_EL12
11	101	0010	0000	001	TTBR1_EL12

op0	op1	CRn	CRm	op2	Mnemonic
11	op1[2:0]	1x11	Cm[3:0]	op2[2:0]	S3_<op1>_C<Cn>_C<Cm>_<op2>

Accessed using MRS/MSR:

op0	op1	CRn	CRm	op2	Access	Mnemonic	
10	000	0000	0000	010	RW	OSDTRRX_EL1	OSDTR
10	000	0000	0010	000	RW	MDCCINT_EL1	MDCC
10	000	0000	0010	010	RW	MDSCR_EL1	MDSC
10	000	0000	0011	010	RW	OSDTRTX_EL1	OSDTR
10	000	0000	0100	010	RW	MDSELR_EL1	MDSE
10	000	0000	0101	010	RW	MDSTEPOP_EL1	MDST
10	000	0000	0110	010	RW	OSECCR_EL1	OSEC
10	000	0000	m[3:0]	100	RW	DBGBVR<m>_EL1	DBGB
10	000	0000	m[3:0]	101	RW	DBGBCR<m>_EL1	DBGB
10	000	0000	m[3:0]	110	RW	DBGWVR<m>_EL1	DBGW
10	000	0000	m[3:0]	111	RW	DBGWCR<m>_EL1	DBGW
10	000	0001	0000	000	RO	MDRAR_EL1	MDRA
10	000	0001	0000	100	WO	OSLAR_EL1	OSLA
10	000	0001	0001	100	RO	OSLSR_EL1	OSLSR
10	000	0001	0011	100	RW	OSDLR_EL1	OSDL
10	000	0001	0100	100	RW	DBGPRCR_EL1	DBGP
10	000	0111	1000	110	RW	DBGCLAIMSET_EL1	DBGC
10	000	0111	1001	110	RW	DBGCLAIMCLR_EL1	DBGC
10	000	0111	1110	110	RO	DBGAUTHSTATUS_EL1	DBGA
10	000	1001	1101	00:m[0]	RO	SPMCGCR<m>_EL1	SPMC
10	000	1001	1101	011	RW	SPMACCESSR_EL1	SPMA
10	000	1001	1101	011	RW	SPMACCESSR_EL1	SPMA
10	000	1001	1101	100	RO	SPMIIDR_EL1	SPMII
10	000	1001	1101	101	RO	SPMDEVARCH_EL1	SPMD
10	000	1001	1101	110	RO	SPMDEVAFF_EL1	SPMD
10	000	1001	1101	111	RO	SPMCFGR_EL1	SPMC
10	000	1001	1110	001	RW	SPMINTENSET_EL1	SPMIN
10	000	1001	1110	010	RW	SPMINTENCLR_EL1	SPMIN
10	000	1110	1011	111	RO	PMCCNTSVR_EL1	PMCC
10	000	1110	10:m[4:3]	m[2:0]	RO	PMEVCNTSVR<m>_EL1	PMEV
10	000	1110	1100	000	RO	PMICNTSVR_EL1	PMICN
10	001	0000	0000	001	RW	TRCTRACEIDR	TRCTR
10	001	0000	0000	010	RW	TRCVICTLR	TRCVI
10	001	0000	0000	110	RO	TRCIDR8	TRCID
10	001	0000	0000	111	RW	TRCIMSPEC0	TRCIM
10	001	0000	0001	000	RW	TRCPRGCTLR	TRCPR
10	001	0000	0001	001	RW	TRCQCTLR	TRCQ
10	001	0000	0001	010	RW	TRCVIIECTLR	TRCVI
10	001	0000	0001	110	RO	TRCIDR9	TRCID
10	001	0000	0010	001	RW	TRCITEEDCR	TRCIT
10	001	0000	0010	010	RW	TRCVISSCTLR	TRCVI
10	001	0000	0010	110	RO	TRCIDR10	TRCID
10	001	0000	0011	000	RO	TRCSTATR	TRCST
10	001	0000	0011	010	RW	TRCVIPCSSCTLR	TRCVI

op0	op1	CRn	CRm	op2	Access	Mnemonic	
10	001	0000	0011	110	RO	TRCIDR11	TRCID
10	001	0000	00:m[1:0]	100	RW	TRCSEQEVR<m>	TRCSE
10	001	0000	00:m[1:0]	101	RW	TRCCNTRLDVR<m>	TRCCN
10	001	0000	0100	000	RW	TRCCONFIGR	TRCC(
10	001	0000	0100	110	RO	TRCIDR12	TRCID
10	001	0000	0101	110	RO	TRCIDR13	TRCID
10	001	0000	0110	000	RW	TRCAUXCTLR	TRCAU
10	001	0000	0110	100	RW	TRCSEQRSTEVR	TRCSE
10	001	0000	0111	100	RW	TRCSEQSTR	TRCSE
10	001	0000	01:m[1:0]	101	RW	TRCCNTCTLR<m>	TRCCN
10	001	0000	0:m[2:0]	111	RW	TRCIMSPEC<m>	TRCIM
10	001	0000	1000	000	RW	TRCEVENTCTL0R	TRCEV
10	001	0000	1000	111	RO	TRCIDR0	TRCID
10	001	0000	1001	000	RW	TRCEVENTCTL1R	TRCEV
10	001	0000	1001	111	RO	TRCIDR1	TRCID
10	001	0000	1010	000	RW	TRCRSR	TRCRS
10	001	0000	1010	111	RO	TRCIDR2	TRCID
10	001	0000	1011	000	RW	TRCSTALLCTLR	TRCST
10	001	0000	1011	111	RO	TRCIDR3	TRCID
10	001	0000	10:m[1:0]	100	RW	TRCEXTINSELR<m>	TRCEX
10	001	0000	10:m[1:0]	101	RW	TRCCNTVR<m>	TRCCN
10	001	0000	1100	000	RW	TRCTSCTLR	TRCTS
10	001	0000	1100	111	RO	TRCIDR4	TRCID
10	001	0000	1101	000	RW	TRCSYNCPR	TRCSY
10	001	0000	1101	111	RO	TRCIDR5	TRCID
10	001	0000	1110	000	RW	TRCCCCTLR	TRCC(
10	001	0000	1110	111	RO	TRCIDR6	TRCID
10	001	0000	1111	000	RW	TRCBBCTLR	TRCBB
10	001	0000	1111	111	RO	TRCIDR7	TRCID
10	001	0001	0001	100	RO	TRCOSLSR	TRCOS
10	001	0001	0:m[2:0]	010	RW	TRCSSCCR<m>	TRCSS
10	001	0001	0:m[2:0]	011	RW	TRCSSPCICR<m>	TRCSS
10	001	0001	1:m[2:0]	010	RW	TRCSSCSR<m>	TRCSS
10	001	0001	m[3:0]	00:m[4]	RW	TRCRSCTLR<m>	TRCRS
10	001	0010	m[2:0]:0	00:m[3]	RW	TRCACVR<m>	TRCAC
10	001	0010	m[2:0]:0	01:m[3]	RW	TRCACATR<m>	TRCAC
10	001	0011	0000	010	RW	TRCCIDCCTLR0	TRCCI
10	001	0011	0001	010	RW	TRCCIDCCTLR1	TRCCI
10	001	0011	0010	010	RW	TRCVMIDCCTLR0	TRCVM
10	001	0011	0011	010	RW	TRCVMIDCCTLR1	TRCVM
10	001	0011	m[2:0]:0	000	RW	TRCCIDCVR<m>	TRCCI
10	001	0011	m[2:0]:0	001	RW	TRCVMIDCVR<m>	TRCVM
10	001	0111	0010	111	RO	TRCDEVID	TRCDE
10	001	0111	1000	110	RW	TRCCLAIMSET	TRCCL
10	001	0111	1001	110	RW	TRCCLAIMCLR	TRCCL
10	001	0111	1110	110	RO	TRCAUTHSTATUS	TRCAU
10	001	0111	1111	110	RO	TRCDEVARCH	TRCDE
10	001	1000	m[3:0]	m[4]:00	RO	BRBINF<m>_EL1	BRBIN
10	001	1000	m[3:0]	m[4]:01	RO	BRBSRC<m>_EL1	BRBSR

op0	op1	CRn	CRm	op2	Access	Mnemonic	
10	001	1000	m[3:0]	m[4]:10	RO	BRBTGT<m>_EL1	BRBTG
10	001	1001	0000	000	RW	BRBCR_EL1	BRBCI
10	001	1001	0000	000	RW	BRBCR_EL1	BRBCI
10	001	1001	0000	001	RW	BRBFCR_EL1	BRBFC
10	001	1001	0000	010	RW	BRBTS_EL1	BRBTS
10	001	1001	0001	000	RW	BRBINFINJ_EL1	BRBIN
10	001	1001	0001	001	RW	BRBSRCINJ_EL1	BRBSF
10	001	1001	0001	010	RW	BRBTGTINJ_EL1	BRBTG
10	001	1001	0010	000	RO	BRBIDR0_EL1	BRBID
10	011	0000	0001	000	RO	MDCCSR_EL0	MDCC
10	011	0000	0100	000	RW	DBGDTR_EL0	DBGD
10	011	0000	0101	000	RO	DBGDTRRX_EL0	DBGD
10	011	0000	0101	000	WO	DBGDTRTX_EL0	DBGD
10	011	1001	1100	000	RW	SPMCR_EL0	SPMCI
10	011	1001	1100	001	RW	SPMCNTENSET_EL0	SPMCN
10	011	1001	1100	010	RW	SPMCNTENCLR_EL0	SPMCN
10	011	1001	1100	011	RW	SPMOVSCLR_EL0	SPMOV
10	011	1001	1100	100	WO	SPMZR_EL0	SPMZI
10	011	1001	1100	101	RW	SPMSELR_EL0	SPMSI
10	011	1001	1110	011	RW	SPMOVSSET_EL0	SPMOV
10	011	1110	000:m[3]	m[2:0]	RW	SPMEVCNTR<m>_EL0	SPMEV
10	011	1110	001:m[3]	m[2:0]	RW	SPMEVTYPER<m>_EL0	SPMEV
10	011	1110	010:m[3]	m[2:0]	RW	SPMEVFILTR<m>_EL0	SPMEV
10	011	1110	011:m[3]	m[2:0]	RW	SPMEVFILT2R<m>_EL0	SPMEV
10	100	0000	0111	000	RW	DBGVCR32_EL2	DBGV
10	100	1001	0000	000	RW	BRBCR_EL2	BRBCI
10	100	1001	1101	011	RW	SPMACCESSR_EL2	SPMAC
10	101	1001	0000	000	RW	BRBCR_EL12	BRBCI
10	101	1001	1101	011	RW	SPMACCESSR_EL12	SPMAC
10	110	1001	1101	011	RW	SPMACCESSR_EL3	SPMAC
10	110	1001	1110	111	RW	SPMROOTCR_EL3	SPMRC
10	111	1001	1110	111	RW	SPMSCR_EL1	SPMSC
11	000	0000	0000	000	RO	MIDR_EL1	MIDR_
11	000	0000	0000	000	RO	MIDR_EL1	VPIDR
11	000	0000	0000	101	RO	MPIDR_EL1	MPIDR
11	000	0000	0000	101	RO	MPIDR_EL1	VMPID
11	000	0000	0000	110	RO	REVIDR_EL1	REVID
11	000	0000	0001	000	RO	ID_PFR0_EL1	ID_PFI
11	000	0000	0001	001	RO	ID_PFR1_EL1	ID_PFI
11	000	0000	0001	010	RO	ID_DFR0_EL1	ID_DF
11	000	0000	0001	011	RO	ID_AFR0_EL1	ID_AF
11	000	0000	0001	100	RO	ID_MMFR0_EL1	ID_MM
11	000	0000	0001	101	RO	ID_MMFR1_EL1	ID_MM
11	000	0000	0001	110	RO	ID_MMFR2_EL1	ID_MM
11	000	0000	0001	111	RO	ID_MMFR3_EL1	ID_MM
11	000	0000	0010	000	RO	ID_ISAR0_EL1	ID_ISA
11	000	0000	0010	001	RO	ID_ISAR1_EL1	ID_ISA
11	000	0000	0010	010	RO	ID_ISAR2_EL1	ID_ISA
11	000	0000	0010	011	RO	ID_ISAR3_EL1	ID_ISA

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	000	0000	0010	100	RO	ID_ISAR4_EL1	ID_ISA
11	000	0000	0010	101	RO	ID_ISAR5_EL1	ID_ISA
11	000	0000	0010	110	RO	ID_MMFR4_EL1	ID_MM
11	000	0000	0010	111	RO	ID_ISAR6_EL1	ID_ISA
11	000	0000	0011	000	RO	MVFR0_EL1	MVFR
11	000	0000	0011	001	RO	MVFR1_EL1	MVFR
11	000	0000	0011	010	RO	MVFR2_EL1	MVFR
11	000	0000	0011	100	RO	ID_PFR2_EL1	ID_PFI
11	000	0000	0011	101	RO	ID_DFR1_EL1	ID_DF
11	000	0000	0011	110	RO	ID_MMFR5_EL1	ID_MM
11	000	0000	0100	000	RO	ID_AA64PFR0_EL1	ID_AA
11	000	0000	0100	001	RO	ID_AA64PFR1_EL1	ID_AA
11	000	0000	0100	010	RO	ID_AA64PFR2_EL1	ID_AA
11	000	0000	0100	100	RO	ID_AA64ZFR0_EL1	ID_AA
11	000	0000	0100	101	RO	ID_AA64SMFR0_EL1	ID_AA
11	000	0000	0100	111	RO	ID_AA64FPFR0_EL1	ID_AA
11	000	0000	0101	000	RO	ID_AA64DFR0_EL1	ID_AA
11	000	0000	0101	001	RO	ID_AA64DFR1_EL1	ID_AA
11	000	0000	0101	010	RO	ID_AA64DFR2_EL1	ID_AA
11	000	0000	0101	100	RO	ID_AA64AFR0_EL1	ID_AA
11	000	0000	0101	101	RO	ID_AA64AFR1_EL1	ID_AA
11	000	0000	0110	000	RO	ID_AA64ISAR0_EL1	ID_AA
11	000	0000	0110	001	RO	ID_AA64ISAR1_EL1	ID_AA
11	000	0000	0110	010	RO	ID_AA64ISAR2_EL1	ID_AA
11	000	0000	0110	011	RO	ID_AA64ISAR3_EL1	ID_AA
11	000	0000	0111	000	RO	ID_AA64MMFR0_EL1	ID_AA
11	000	0000	0111	001	RO	ID_AA64MMFR1_EL1	ID_AA
11	000	0000	0111	010	RO	ID_AA64MMFR2_EL1	ID_AA
11	000	0000	0111	011	RO	ID_AA64MMFR3_EL1	ID_AA
11	000	0000	0111	100	RO	ID_AA64MMFR4_EL1	ID_AA
11	000	0001	0000	000	RW	SCTLR_EL1	SCTLR
11	000	0001	0000	000	RW	SCTLR_EL1	SCTLR
11	000	0001	0000	001	RW	ACTLR_EL1	ACTLI
11	000	0001	0000	001	RW	ACTLR_EL1	ACTLI
11	000	0001	0000	010	RW	CPACR_EL1	CPACI
11	000	0001	0000	010	RW	CPACR_EL1	CPTR_
11	000	0001	0000	011	RW	SCTLR2_EL1	SCTLR
11	000	0001	0000	011	RW	SCTLR2_EL1	SCTLR
11	000	0001	0000	101	RW	RGSR_EL1	RGSR_
11	000	0001	0000	110	RW	GCR_EL1	GCR_E
11	000	0001	0010	000	RW	ZCR_EL1	ZCR_E
11	000	0001	0010	000	RW	ZCR_EL1	ZCR_E
11	000	0001	0010	001	RW	TRFCR_EL1	TRFCR
11	000	0001	0010	001	RW	TRFCR_EL1	TRFCR
11	000	0001	0010	011	RW	TRCITECR_EL1	TRCIT
11	000	0001	0010	011	RW	TRCITECR_EL1	TRCIT
11	000	0001	0010	100	RW	SMPRI_EL1	SMPRI
11	000	0001	0010	110	RW	SMCR_EL1	SMCR_
11	000	0001	0010	110	RW	SMCR_EL1	SMCR_

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	000	0001	0100	000	RW	SCTLRMASK_EL1	SCTLR
11	000	0001	0100	000	RW	SCTLRMASK_EL1	SCTLR
11	000	0001	0100	001	RW	ACTLRMASK_EL1	ACTLR
11	000	0001	0100	001	RW	ACTLRMASK_EL1	ACTLR
11	000	0001	0100	010	RW	CPACRMASK_EL1	CPACR
11	000	0001	0100	010	RW	CPACRMASK_EL1	CPTRM
11	000	0001	0100	011	RW	SCTLR2MASK_EL1	SCTLR
11	000	0001	0100	011	RW	SCTLR2MASK_EL1	SCTLR
11	000	0001	0100	100	RW	CPACRALIAS_EL1	CPACR
11	000	0001	0100	101	RW	ACTLRALIAS_EL1	ACTLR
11	000	0001	0100	110	RW	SCTLRALIAS_EL1	SCTLR
11	000	0001	0100	111	RW	SCTLR2ALIAS_EL1	SCTLR
11	000	0010	0000	000	RW	TTBR0_EL1	TTBR0
11	000	0010	0000	000	RW	TTBR0_EL1	TTBR0
11	000	0010	0000	001	RW	TTBR1_EL1	TTBR1
11	000	0010	0000	001	RW	TTBR1_EL1	TTBR1
11	000	0010	0000	010	RW	TCR_EL1	TCR_E
11	000	0010	0000	010	RW	TCR_EL1	TCR_E
11	000	0010	0000	011	RW	TCR2_EL1	TCR2_
11	000	0010	0000	011	RW	TCR2_EL1	TCR2_
11	000	0010	0001	000	RW	APIAKeyLo_EL1	APIAK
11	000	0010	0001	001	RW	APIAKeyHi_EL1	APIAK
11	000	0010	0001	010	RW	APIBKeyLo_EL1	APIBK
11	000	0010	0001	011	RW	APIBKeyHi_EL1	APIBK
11	000	0010	0010	000	RW	APDAKeyLo_EL1	APDAI
11	000	0010	0010	001	RW	APDAKeyHi_EL1	APDAI
11	000	0010	0010	010	RW	APDBKeyLo_EL1	APDBI
11	000	0010	0010	011	RW	APDBKeyHi_EL1	APDBI
11	000	0010	0011	000	RW	APGAKeyLo_EL1	APGAI
11	000	0010	0011	001	RW	APGAKeyHi_EL1	APGAI
11	000	0010	0101	000	RW	GCSCR_EL1	GCSCI
11	000	0010	0101	000	RW	GCSCR_EL1	GCSCI
11	000	0010	0101	001	RW	GCSPR_EL1	GCSPI
11	000	0010	0101	001	RW	GCSPR_EL1	GCSPI
11	000	0010	0101	010	RW	GCSCRE0_EL1	GCSCI
11	000	0010	0111	010	RW	TCRMASK_EL1	TCRM
11	000	0010	0111	010	RW	TCRMASK_EL1	TCRM
11	000	0010	0111	011	RW	TCR2MASK_EL1	TCR2M
11	000	0010	0111	011	RW	TCR2MASK_EL1	TCR2M
11	000	0010	0111	110	RW	TCRALIAS_EL1	TCR_E
11	000	0010	0111	111	RW	TCR2ALIAS_EL1	TCR2_
11	000	0100	0000	000	RW	SPSR_EL1	SPSR_
11	000	0100	0000	000	RW	SPSR_EL1	SPSR_
11	000	0100	0000	001	RW	ELR_EL1	ELR_E
11	000	0100	0000	001	RW	ELR_EL1	ELR_E
11	000	0100	0001	000	RW	SP_EL0	SP_EL
11	000	0100	0010	000	RW	SPSel	SPSel
11	000	0100	0010	010	RO	CurrentEL	Current
11	000	0100	0010	011	RW	PAN	PAN

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	000	0100	0010	100	RW	UAO	UAO
11	000	0100	0011	000	RW	ALLINT	ALLIN
11	000	0100	0011	001	RW	PM	PM
11	000	0100	0110	000	RW	ICC_PMR_EL1	ICC_P
11	000	0100	0110	000	RW	ICC_PMR_EL1	ICV_P
11	000	0101	0001	000	RW	AFSR0_EL1	AFSR0
11	000	0101	0001	000	RW	AFSR0_EL1	AFSR0
11	000	0101	0001	001	RW	AFSR1_EL1	AFSR1
11	000	0101	0001	001	RW	AFSR1_EL1	AFSR1
11	000	0101	0010	000	RW	ESR_EL1	ESR_E
11	000	0101	0010	000	RW	ESR_EL1	ESR_E
11	000	0101	0011	000	RO	ERRIDR_EL1	ERRID
11	000	0101	0011	001	RW	ERRSELR_EL1	ERRSE
11	000	0101	0011	010	RO	ERXGSR_EL1	ERXGS
11	000	0101	0100	000	RO	ERXFR_EL1	ERXFR
11	000	0101	0100	001	RW	ERXCTLR_EL1	ERXCT
11	000	0101	0100	010	RW	ERXSTATUS_EL1	ERXST
11	000	0101	0100	011	RW	ERXADDR_EL1	ERXAD
11	000	0101	0100	100	RO	ERXPFGF_EL1	ERXPF
11	000	0101	0100	101	RW	ERXPFGCTL_EL1	ERXPF
11	000	0101	0100	110	RW	ERXPFGCDN_EL1	ERXPF
11	000	0101	0101	000	RW	ERXMISC0_EL1	ERXM
11	000	0101	0101	001	RW	ERXMISC1_EL1	ERXM
11	000	0101	0101	010	RW	ERXMISC2_EL1	ERXM
11	000	0101	0101	011	RW	ERXMISC3_EL1	ERXM
11	000	0101	0110	000	RW	TFSR_EL1	TFSR_
11	000	0101	0110	000	RW	TFSR_EL1	TFSR_
11	000	0101	0110	001	RW	TFSRE0_EL1	TFSRE
11	000	0110	0000	000	RW	FAR_EL1	FAR_E
11	000	0110	0000	000	RW	FAR_EL1	FAR_E
11	000	0110	0000	101	RW	PFAR_EL1	PFAR_
11	000	0111	0100	000	RW	PAR_EL1	PAR_E
11	000	1001	1001	000	RW	PMSCR_EL1	PMSCR
11	000	1001	1001	000	RW	PMSCR_EL1	PMSCR
11	000	1001	1001	001	RW	PMSNEVFR_EL1	PMSNE
11	000	1001	1001	010	RW	PMSICR_EL1	PMSIC
11	000	1001	1001	011	RW	PMSIRR_EL1	PMSIR
11	000	1001	1001	100	RW	PMSFCR_EL1	PMSFC
11	000	1001	1001	101	RW	PMSEVFR_EL1	PMSEV
11	000	1001	1001	110	RW	PMSLATFR_EL1	PMSLA
11	000	1001	1001	111	RO	PMSIDR_EL1	PMSID
11	000	1001	1010	000	RW	PMBLIMITR_EL1	PMBLI
11	000	1001	1010	001	RW	PMBPTR_EL1	PMBPT
11	000	1001	1010	011	RW	PMBSR_EL1	PMBSR
11	000	1001	1010	011	RW	PMBSR_EL1	PMBSR
11	000	1001	1010	100	RW	PMSDSFR_EL1	PMSDS
11	000	1001	1010	101	RW	PMBMAR_EL1	PMBMA
11	000	1001	1010	111	RO	PMBIDR_EL1	PMBID
11	000	1001	1011	000	RW	TRBLIMITR_EL1	TRBLI

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	000	1001	1011	001	RW	TRBPTR_EL1	TRBPT
11	000	1001	1011	010	RW	TRBBASER_EL1	TRBBA
11	000	1001	1011	011	RW	TRBSR_EL1	TRBSR
11	000	1001	1011	011	RW	TRBSR_EL1	TRBSR
11	000	1001	1011	100	RW	TRBMAR_EL1	TRBM
11	000	1001	1011	101	RW	TRBMPAM_EL1	TRBM
11	000	1001	1011	110	RW	TRBTRG_EL1	TRBTR
11	000	1001	1011	111	RO	TRBIDR_EL1	TRBID
11	000	1001	1101	011	RW	PMSSCR_EL1	PMSSC
11	000	1001	1110	001	RW	PMINTENSET_EL1	PMINT
11	000	1001	1110	010	RW	PMINTENCLR_EL1	PMINT
11	000	1001	1110	100	RW	PMUACR_EL1	PMUA
11	000	1001	1110	101	RW	PMECR_EL1	PMEC
11	000	1001	1110	110	RO	PMMIR_EL1	PMMI
11	000	1001	1110	111	RW	PMIAR_EL1	PMIAR
11	000	1010	0010	000	RW	MAIR_EL1	MAIR_
11	000	1010	0010	000	RW	MAIR_EL1	MAIR_
11	000	1010	0010	001	RW	MAIR2_EL1	MAIR2
11	000	1010	0010	001	RW	MAIR2_EL1	MAIR2
11	000	1010	0010	010	RW	PIRE0_EL1	PIRE0_
11	000	1010	0010	010	RW	PIRE0_EL1	PIRE0_
11	000	1010	0010	011	RW	PIR_EL1	PIR_EL
11	000	1010	0010	011	RW	PIR_EL1	PIR_EL
11	000	1010	0010	100	RW	POR_EL1	POR_E
11	000	1010	0010	100	RW	POR_EL1	POR_E
11	000	1010	0010	101	RW	S2POR_EL1	S2POR
11	000	1010	0011	000	RW	AMAIR_EL1	AMAIR
11	000	1010	0011	000	RW	AMAIR_EL1	AMAIR
11	000	1010	0011	001	RW	AMAIR2_EL1	AMAIR
11	000	1010	0011	001	RW	AMAIR2_EL1	AMAIR
11	000	1010	0100	000	RW	LORSA_EL1	LORSA
11	000	1010	0100	001	RW	LOREA_EL1	LOREA
11	000	1010	0100	010	RW	LORN_EL1	LORN_
11	000	1010	0100	011	RW	LORC_EL1	LORC_
11	000	1010	0100	100	RO	MPAMIDR_EL1	MPAM
11	000	1010	0100	101	RO	MPAMBWIDR_EL1	MPAM
11	000	1010	0100	111	RO	LORID_EL1	LORID
11	000	1010	0101	000	RW	MPAM1_EL1	MPAM
11	000	1010	0101	000	RW	MPAM1_EL1	MPAM
11	000	1010	0101	001	RW	MPAM0_EL1	MPAM
11	000	1010	0101	011	RW	MPAMSM_EL1	MPAM
11	000	1010	0101	100	RW	MPAMBW1_EL1	MPAM
11	000	1010	0101	100	RW	MPAMBW1_EL1	MPAM
11	000	1010	0101	101	RW	MPAMBW0_EL1	MPAM
11	000	1010	0101	111	RW	MPAMBWSM_EL1	MPAM
11	000	1100	0000	000	RW	VBAR_EL1	VBAR_
11	000	1100	0000	000	RW	VBAR_EL1	VBAR_
11	000	1100	0000	001	RO	RVBAR_EL1	RVBA
11	000	1100	0000	010	RW	RMR_EL1	RMR_

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	000	1100	0001	000	RO	ISR_EL1	ISR_EL
11	000	1100	0001	001	RW	DISR_EL1	DISR_
11	000	1100	0001	001	RW	DISR_EL1	VDISR
11	000	1100	0001	001	RW	DISR_EL1	VDISR
11	000	1100	1000	000	RO	ICC_IAR0_EL1	ICC_IA
11	000	1100	1000	000	RO	ICC_IAR0_EL1	ICV_IA
11	000	1100	1000	001	WO	ICC_EOIR0_EL1	ICC_E
11	000	1100	1000	001	WO	ICC_EOIR0_EL1	ICV_E
11	000	1100	1000	010	RO	ICC_HPPIR0_EL1	ICC_H
11	000	1100	1000	010	RO	ICC_HPPIR0_EL1	ICV_H
11	000	1100	1000	011	RW	ICC_BPR0_EL1	ICC_B
11	000	1100	1000	011	RW	ICC_BPR0_EL1	ICV_B
11	000	1100	1000	1:m[1:0]	RW	ICC_AP0R<m>_EL1	ICC_A
11	000	1100	1000	1:m[1:0]	RW	ICC_AP0R<m>_EL1	ICV_A
11	000	1100	1001	0:m[1:0]	RW	ICC_AP1R<m>_EL1	ICC_A
11	000	1100	1001	0:m[1:0]	RW	ICC_AP1R<m>_EL1	ICV_A
11	000	1100	1001	101	RO	ICC_NMIAR1_EL1	ICC_N
11	000	1100	1001	101	RO	ICC_NMIAR1_EL1	ICV_N
11	000	1100	1011	001	WO	ICC_DIR_EL1	ICC_D
11	000	1100	1011	001	WO	ICC_DIR_EL1	ICV_D
11	000	1100	1011	011	RO	ICC_RPR_EL1	ICC_R
11	000	1100	1011	011	RO	ICC_RPR_EL1	ICV_R
11	000	1100	1011	101	WO	ICC_SGI1R_EL1	ICC_S
11	000	1100	1011	110	WO	ICC_ASGI1R_EL1	ICC_A
11	000	1100	1011	111	WO	ICC_SGI0R_EL1	ICC_S
11	000	1100	1100	000	RO	ICC_IAR1_EL1	ICC_IA
11	000	1100	1100	000	RO	ICC_IAR1_EL1	ICV_IA
11	000	1100	1100	001	WO	ICC_EOIR1_EL1	ICC_E
11	000	1100	1100	001	WO	ICC_EOIR1_EL1	ICV_E
11	000	1100	1100	010	RO	ICC_HPPIR1_EL1	ICC_H
11	000	1100	1100	010	RO	ICC_HPPIR1_EL1	ICV_H
11	000	1100	1100	011	RW	ICC_BPR1_EL1	ICC_B
11	000	1100	1100	011	RW	ICC_BPR1_EL1	ICV_B
11	000	1100	1100	100	RW	ICC_CTLR_EL1	ICC_C
11	000	1100	1100	100	RW	ICC_CTLR_EL1	ICV_C
11	000	1100	1100	101	RW	ICC_SRE_EL1	ICC_SI
11	000	1100	1100	110	RW	ICC_IGRPEN0_EL1	ICC_IG
11	000	1100	1100	110	RW	ICC_IGRPEN0_EL1	ICV_IG
11	000	1100	1100	111	RW	ICC_IGRPEN1_EL1	ICC_IG
11	000	1100	1100	111	RW	ICC_IGRPEN1_EL1	ICV_IG
11	000	1101	0000	001	RW	CONTEXTIDR_EL1	CONTI
11	000	1101	0000	001	RW	CONTEXTIDR_EL1	CONTI
11	000	1101	0000	011	RW	RCWSMASK_EL1	RCWS
11	000	1101	0000	100	RW	TPIDR_EL1	TPIDR
11	000	1101	0000	101	RW	ACCDATA_EL1	ACCDA
11	000	1101	0000	110	RW	RCWMASK_EL1	RCWM
11	000	1101	0000	111	RW	SCXTNUM_EL1	SCXTN
11	000	1101	0000	111	RW	SCXTNUM_EL1	SCXTN
11	000	1110	0001	000	RW	CNTKCTL_EL1	CNTHC

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	000	1110	0001	000	RW	CNTKCTL_EL1	CNTK
11	001	0000	0000	000	RO	CCSIDR_EL1	CCSID
11	001	0000	0000	001	RO	CLIDR_EL1	CLIDR
11	001	0000	0000	010	RO	CCSIDR2_EL1	CCSID
11	001	0000	0000	100	RO	GMID_EL1	GMID_
11	001	0000	0000	110	RO	SMIDR_EL1	SMIDR
11	001	0000	0000	111	RO	AIDR_EL1	AIDR_
11	010	0000	0000	000	RW	CSSELR_EL1	CSSEL
11	011	0000	0000	001	RO	CTR_EL0	CTR_E
11	011	0000	0000	111	RO	DCZID_EL0	DCZID
11	011	0010	0100	000	RO	RNDR	RNDR
11	011	0010	0100	001	RO	RNDRRS	RNDR
11	011	0010	0101	001	RW	GCSPR_EL0	GCSPR
11	011	0100	0010	000	RW	NZCV	NZCV
11	011	0100	0010	001	RW	DAIF	DAIF
11	011	0100	0010	010	RW	SVCR	SVCR
11	011	0100	0010	101	RW	DIT	DIT
11	011	0100	0010	110	RW	SSBS	SSBS
11	011	0100	0010	111	RW	TCO	TCO
11	011	0100	0100	000	RW	FPCR	FPCR
11	011	0100	0100	001	RW	FPSR	FPSR
11	011	0100	0100	010	RW	FPMR	FPMR
11	011	0100	0101	000	RW	DSPSR_EL0	DSPSR
11	011	0100	0101	001	RW	DLR_EL0	DLR_E
11	011	1001	0100	000	RW	PMICNTR_EL0	PMICN
11	011	1001	0110	000	RW	PMICFILTR_EL0	PMICF
11	011	1001	1100	000	RW	PMCR_EL0	PMCR
11	011	1001	1100	001	RW	PMCNTENSET_EL0	PMCN
11	011	1001	1100	010	RW	PMCNTENCLR_EL0	PMCN
11	011	1001	1100	011	RW	PMOVSCLR_EL0	PMOV
11	011	1001	1100	100	WO	PMSWINC_EL0	PMSW
11	011	1001	1100	101	RW	PMSELR_EL0	PMSEL
11	011	1001	1100	110	RO	PMCEID0_EL0	PMCEI
11	011	1001	1100	111	RO	PMCEID1_EL0	PMCEI
11	011	1001	1101	000	RW	PMCCNTR_EL0	PMCC
11	011	1001	1101	001	RW	PMXEVTYPER_EL0	PMXE
11	011	1001	1101	010	RW	PMXEVCNTR_EL0	PMXE
11	011	1001	1101	100	WO	PMZR_EL0	PMZR_
11	011	1001	1110	000	RW	PMUSERENR_EL0	PMUSE
11	011	1001	1110	011	RW	PMOVSSET_EL0	PMOV
11	011	1010	0010	100	RW	POR_EL0	POR_E
11	011	1101	0000	010	RW	TPIDR_EL0	TPIDR
11	011	1101	0000	011	RW	TPIDRRO_EL0	TPIDR
11	011	1101	0000	101	RW	TPIDR2_EL0	TPIDR
11	011	1101	0000	111	RW	SCXTNUM_EL0	SCXTN
11	011	1101	0010	000	RW	AMCR_EL0	AMCR
11	011	1101	0010	001	RO	AMCFGR_EL0	AMCF
11	011	1101	0010	010	RO	AMCGCR_EL0	AMCG
11	011	1101	0010	011	RW	AMUSERENR_EL0	AMUS

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	011	1101	0010	100	RW	AMCNTENCLR0_EL0	AMCN
11	011	1101	0010	101	RW	AMCNTENSET0_EL0	AMCN
11	011	1101	0010	110	RO	AMCG1IDR_EL0	AMCG
11	011	1101	0011	000	RW	AMCNTENCLR1_EL0	AMCN
11	011	1101	0011	001	RW	AMCNTENSET1_EL0	AMCN
11	011	1101	010:m[3]	m[2:0]	RW	AMEVCNTR0<m>_EL0	AMEV
11	011	1101	011:m[3]	m[2:0]	RO	AMEVTYPER0<m>_EL0	AMEV
11	011	1101	110:m[3]	m[2:0]	RW	AMEVCNTR1<m>_EL0	AMEV
11	011	1101	111:m[3]	m[2:0]	RW	AMEVTYPER1<m>_EL0	AMEV
11	011	1110	0000	000	RW	CNTFRQ_EL0	CNTF
11	011	1110	0000	001	RO	CNTPCT_EL0	CNTP
11	011	1110	0000	010	RO	CNTVCT_EL0	CNTV
11	011	1110	0000	101	RO	CNTPCTSS_EL0	CNTP
11	011	1110	0000	110	RO	CNTVCTSS_EL0	CNTV
11	011	1110	0010	000	RW	CNTP_TVAL_EL0	CNTH
11	011	1110	0010	000	RW	CNTP_TVAL_EL0	CNTH
11	011	1110	0010	000	RW	CNTP_TVAL_EL0	CNTP_
11	011	1110	0010	001	RW	CNTP_CTL_EL0	CNTH
11	011	1110	0010	001	RW	CNTP_CTL_EL0	CNTH
11	011	1110	0010	001	RW	CNTP_CTL_EL0	CNTP_
11	011	1110	0010	010	RW	CNTP_CVAL_EL0	CNTH
11	011	1110	0010	010	RW	CNTP_CVAL_EL0	CNTH
11	011	1110	0010	010	RW	CNTP_CVAL_EL0	CNTP_
11	011	1110	0011	000	RW	CNTV_TVAL_EL0	CNTH
11	011	1110	0011	000	RW	CNTV_TVAL_EL0	CNTH
11	011	1110	0011	000	RW	CNTV_TVAL_EL0	CNTV_
11	011	1110	0011	001	RW	CNTV_CTL_EL0	CNTH
11	011	1110	0011	001	RW	CNTV_CTL_EL0	CNTH
11	011	1110	0011	001	RW	CNTV_CTL_EL0	CNTV_
11	011	1110	0011	010	RW	CNTV_CVAL_EL0	CNTH
11	011	1110	0011	010	RW	CNTV_CVAL_EL0	CNTH
11	011	1110	0011	010	RW	CNTV_CVAL_EL0	CNTV_
11	011	1110	10:m[4:3]	m[2:0]	RW	PMEVCNTR<m>_EL0	PMEV
11	011	1110	1111	111	RW	PMCCFILTR_EL0	PMCC
11	011	1110	11:m[4:3]	m[2:0]	RW	PMEVTYPER<m>_EL0	PMEV
11	100	0000	0000	000	RW	VPIDR_EL2	VPIDR
11	100	0000	0000	101	RW	VMPIDR_EL2	VMPID
11	100	0001	0000	000	RW	SCTLR_EL2	SCTLR
11	100	0001	0000	001	RW	ACTLR_EL2	ACTLR
11	100	0001	0000	011	RW	SCTLR2_EL2	SCTLR
11	100	0001	0001	000	RW	HCR_EL2	HCR_E
11	100	0001	0001	001	RW	MDCR_EL2	MDCR
11	100	0001	0001	010	RW	CPTR_EL2	CPTR_
11	100	0001	0001	011	RW	HSTR_EL2	HSTR_
11	100	0001	0001	100	RW	HFGRTR_EL2	HFGRT
11	100	0001	0001	101	RW	HFGWTR_EL2	HFGW
11	100	0001	0001	110	RW	HFGITR_EL2	HFGIT
11	100	0001	0001	111	RW	HACR_EL2	HACR_
11	100	0001	0010	000	RW	ZCR_EL2	ZCR_E

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	100	0001	0010	001	RW	TRFCR_EL2	TRFCR
11	100	0001	0010	010	RW	HCRX_EL2	HCRX
11	100	0001	0010	011	RW	TRCITECR_EL2	TRCIT
11	100	0001	0010	101	RW	SMPRIMAP_EL2	SMPRI
11	100	0001	0010	110	RW	SMCR_EL2	SMCR
11	100	0001	0011	001	RW	SDER32_EL2	SDER3
11	100	0001	0100	000	RW	SCTLRMASK_EL2	SCTLR
11	100	0001	0100	001	RW	ACTLRMASK_EL2	ACTLR
11	100	0001	0100	010	RW	CPTRMASK_EL2	CPTRM
11	100	0001	0100	011	RW	SCTLR2MASK_EL2	SCTLR
11	100	0010	0000	000	RW	TTBR0_EL2	TTBR0
11	100	0010	0000	001	RW	TTBR1_EL2	TTBR1
11	100	0010	0000	010	RW	TCR_EL2	TCR_E
11	100	0010	0000	011	RW	TCR2_EL2	TCR2_
11	100	0010	0001	000	RW	VTTBR_EL2	VTTBR
11	100	0010	0001	010	RW	VTCR_EL2	VTCR_
11	100	0010	0010	000	RW	VNCR_EL2	VNCR_
11	100	0010	0011	010	RW	HDBSSBR_EL2	HDBSS
11	100	0010	0011	011	RW	HDBSSPROD_EL2	HDBSS
11	100	0010	0011	100	RW	HACDBSBR_EL2	HACDB
11	100	0010	0011	101	RW	HACDBSCONS_EL2	HACDB
11	100	0010	0101	000	RW	GCSCR_EL2	GCSCR
11	100	0010	0101	001	RW	GCSPR_EL2	GCSPR
11	100	0010	0110	000	RW	VSTTBR_EL2	VSTTB
11	100	0010	0110	010	RW	VSTCR_EL2	VSTCR
11	100	0010	0111	010	RW	TCRMASK_EL2	TCRM
11	100	0010	0111	011	RW	TCR2MASK_EL2	TCR2M
11	100	0011	0000	000	RW	DACR32_EL2	DACR
11	100	0011	0001	000	RW	HDFGRTR2_EL2	HDFGR
11	100	0011	0001	001	RW	HDFGWTR2_EL2	HDFGW
11	100	0011	0001	010	RW	HFGRTR2_EL2	HFGRT
11	100	0011	0001	011	RW	HFGWTR2_EL2	HFGW
11	100	0011	0001	100	RW	HDFGRTR_EL2	HDFGR
11	100	0011	0001	101	RW	HDFGWTR_EL2	HDFGW
11	100	0011	0001	110	RW	HAFGRTR_EL2	HAFGR
11	100	0011	0001	111	RW	HFGITR2_EL2	HFGIT
11	100	0100	0000	000	RW	SPSR_EL2	SPSR_
11	100	0100	0000	000	RW	SPSR_EL2	SPSR_
11	100	0100	0000	001	RW	ELR_EL2	ELR_E
11	100	0100	0000	001	RW	ELR_EL2	ELR_E
11	100	0100	0001	000	RW	SP_EL1	SP_EL
11	100	0100	0011	000	RW	SPSR_irq	SPSR_
11	100	0100	0011	001	RW	SPSR_abt	SPSR_
11	100	0100	0011	010	RW	SPSR_und	SPSR_
11	100	0100	0011	011	RW	SPSR_fiq	SPSR_
11	100	0101	0000	001	RW	IFSR32_EL2	IFSR32
11	100	0101	0001	000	RW	AFSR0_EL2	AFSR0
11	100	0101	0001	001	RW	AFSR1_EL2	AFSR1
11	100	0101	0010	000	RW	ESR_EL2	ESR_E

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	100	0101	0010	000	RW	ESR_EL2	ESR_E
11	100	0101	0010	011	RW	VSESR_EL2	VSESR
11	100	0101	0011	000	RW	FPEXC32_EL2	FPEXC
11	100	0101	0110	000	RW	TFSR_EL2	TFSR_
11	100	0101	0110	000	RW	TFSR_EL2	TFSR_
11	100	0110	0000	000	RW	FAR_EL2	FAR_E
11	100	0110	0000	000	RW	FAR_EL2	FAR_E
11	100	0110	0000	100	RW	HPFAR_EL2	HPFAR
11	100	0110	0000	101	RW	PFAR_EL2	PFAR_
11	100	1001	1001	000	RW	PMSCR_EL2	PMSCR
11	100	1001	1010	011	RW	PMBSR_EL2	PMBSR
11	100	1001	1011	011	RW	TRBSR_EL2	TRBSR
11	100	1010	0001	001	RW	MAIR2_EL2	MAIR2
11	100	1010	0010	000	RW	MAIR_EL2	MAIR_
11	100	1010	0010	010	RW	PIRE0_EL2	PIRE0_
11	100	1010	0010	011	RW	PIR_EL2	PIR_EL
11	100	1010	0010	100	RW	POR_EL2	POR_E
11	100	1010	0010	101	RW	S2PIR_EL2	S2PIR_
11	100	1010	0011	000	RW	AMAIR_EL2	AMAIR
11	100	1010	0011	001	RW	AMAIR2_EL2	AMAIR
11	100	1010	0100	000	RW	MPAMHCR_EL2	MPAM
11	100	1010	0100	001	RW	MPAMVPMV_EL2	MPAM
11	100	1010	0101	000	RW	MPAM2_EL2	MPAM
11	100	1010	0101	100	RW	MPAMBW2_EL2	MPAM
11	100	1010	0101	110	RW	MPAMBWCAP_EL2	MPAM
11	100	1010	0110	000	RW	MPAMVPM0_EL2	MPAM
11	100	1010	0110	001	RW	MPAMVPM1_EL2	MPAM
11	100	1010	0110	010	RW	MPAMVPM2_EL2	MPAM
11	100	1010	0110	011	RW	MPAMVPM3_EL2	MPAM
11	100	1010	0110	100	RW	MPAMVPM4_EL2	MPAM
11	100	1010	0110	101	RW	MPAMVPM5_EL2	MPAM
11	100	1010	0110	110	RW	MPAMVPM6_EL2	MPAM
11	100	1010	0110	111	RW	MPAMVPM7_EL2	MPAM
11	100	1010	1000	000	RW	MECID_P0_EL2	MECID
11	100	1010	1000	001	RW	MECID_A0_EL2	MECID
11	100	1010	1000	010	RW	MECID_P1_EL2	MECID
11	100	1010	1000	011	RW	MECID_A1_EL2	MECID
11	100	1010	1000	111	RO	MECIDR_EL2	MECID
11	100	1010	1001	000	RW	VMECID_P_EL2	VMECI
11	100	1010	1001	001	RW	VMECID_A_EL2	VMECI
11	100	1100	0000	000	RW	VBAR_EL2	VBAR_
11	100	1100	0000	001	RO	RVBAR_EL2	RVBAR
11	100	1100	0000	010	RW	RMR_EL2	RMR_E
11	100	1100	0001	001	RW	VDISR_EL2	VDISR
11	100	1100	1000	0:m[1:0]	RW	ICH_AP0R<m>_EL2	ICH_A
11	100	1100	1001	0:m[1:0]	RW	ICH_AP1R<m>_EL2	ICH_A
11	100	1100	1001	101	RW	ICC_SRE_EL2	ICC_SR
11	100	1100	1011	000	RW	ICH_HCR_EL2	ICH_H
11	100	1100	1011	001	RO	ICH_VTR_EL2	ICH_V

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	100	1100	1011	010	RO	ICH_MISR_EL2	ICH_M
11	100	1100	1011	011	RO	ICH_EISR_EL2	ICH_E
11	100	1100	1011	101	RO	ICH_ELRSR_EL2	ICH_E
11	100	1100	1011	111	RW	ICH_VMCR_EL2	ICH_V
11	100	1100	110:m[3]	m[2:0]	RW	ICH_LR<m>_EL2	ICH_L
11	100	1101	0000	001	RW	CONTEXTIDR_EL2	CONTI
11	100	1101	0000	010	RW	TPIDR_EL2	TPIDR
11	100	1101	0000	111	RW	SCXTNUM_EL2	SCXTN
11	100	1101	100:m[3]	m[2:0]	RW	AMEVCNTVOFF0<m>_EL2	AMEV
11	100	1101	101:m[3]	m[2:0]	RW	AMEVCNTVOFF1<m>_EL2	AMEV
11	100	1110	0000	011	RW	CNTVOFF_EL2	CNTV
11	100	1110	0000	110	RW	CNTPOFF_EL2	CNTP
11	100	1110	0001	000	RW	CNTHCTL_EL2	CNTH
11	100	1110	0010	000	RW	CNTHP_TVAL_EL2	CNTH
11	100	1110	0010	001	RW	CNTHP_CTL_EL2	CNTH
11	100	1110	0010	010	RW	CNTHP_CVAL_EL2	CNTH
11	100	1110	0011	000	RW	CNTHV_TVAL_EL2	CNTH
11	100	1110	0011	001	RW	CNTHV_CTL_EL2	CNTH
11	100	1110	0011	010	RW	CNTHV_CVAL_EL2	CNTH
11	100	1110	0100	000	RW	CNTHVS_TVAL_EL2	CNTH
11	100	1110	0100	001	RW	CNTHVS_CTL_EL2	CNTH
11	100	1110	0100	010	RW	CNTHVS_CVAL_EL2	CNTH
11	100	1110	0101	000	RW	CNTHPS_TVAL_EL2	CNTH
11	100	1110	0101	001	RW	CNTHPS_CTL_EL2	CNTH
11	100	1110	0101	010	RW	CNTHPS_CVAL_EL2	CNTH
11	101	0001	0000	000	RW	SCTLR_EL12	SCTLR
11	101	0001	0000	001	RW	ACTLR_EL12	ACTLR
11	101	0001	0000	010	RW	CPACR_EL12	CPACR
11	101	0001	0000	011	RW	SCTLR2_EL12	SCTLR
11	101	0001	0010	000	RW	ZCR_EL12	ZCR_E
11	101	0001	0010	001	RW	TRFCR_EL12	TRFCR
11	101	0001	0010	011	RW	TRCITECR_EL12	TRCIT
11	101	0001	0010	110	RW	SMCR_EL12	SMCR_
11	101	0001	0100	000	RW	SCTLRMASK_EL12	SCTLR
11	101	0001	0100	001	RW	ACTLRMASK_EL12	ACTLR
11	101	0001	0100	010	RW	CPACRMASK_EL12	CPACR
11	101	0001	0100	011	RW	SCTLR2MASK_EL12	SCTLR
11	101	0010	0000	000	RW	TTBR0_EL12	TTBR0
11	101	0010	0000	001	RW	TTBR1_EL12	TTBR1
11	101	0010	0000	010	RW	TCR_EL12	TCR_E
11	101	0010	0000	011	RW	TCR2_EL12	TCR2_
11	101	0010	0101	000	RW	GCSCR_EL12	GCSCR
11	101	0010	0101	001	RW	GCSPR_EL12	GCSPR
11	101	0010	0111	010	RW	TCRMASK_EL12	TCRM
11	101	0010	0111	011	RW	TCR2MASK_EL12	TCR2M
11	101	0100	0000	000	RW	SPSR_EL12	SPSR_
11	101	0100	0000	001	RW	ELR_EL12	ELR_E
11	101	0101	0001	000	RW	AFSR0_EL12	AFSR0
11	101	0101	0001	001	RW	AFSR1_EL12	AFSR1

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	101	0101	0010	000	RW	ESR_EL12	ESR_E
11	101	0101	0110	000	RW	TFSR_EL12	TFSR_
11	101	0110	0000	000	RW	FAR_EL12	FAR_E
11	101	0110	0000	101	RW	PFAR_EL12	PFAR_
11	101	1001	1001	000	RW	PMSCR_EL12	PMSCI
11	101	1001	1010	011	RW	PMBSR_EL12	PMBSI
11	101	1001	1011	011	RW	TRBSR_EL12	TRBSI
11	101	1010	0010	000	RW	MAIR_EL12	MAIR_
11	101	1010	0010	001	RW	MAIR2_EL12	MAIR2
11	101	1010	0010	010	RW	PIRE0_EL12	PIRE0
11	101	1010	0010	011	RW	PIR_EL12	PIR_E
11	101	1010	0010	100	RW	POR_EL12	POR_E
11	101	1010	0011	000	RW	AMAIR_EL12	AMAI
11	101	1010	0011	001	RW	AMAIR2_EL12	AMAI
11	101	1010	0101	000	RW	MPAM1_EL12	MPAM
11	101	1010	0101	100	RW	MPAMBW1_EL12	MPAM
11	101	1100	0000	000	RW	VBAR_EL12	VBAR_
11	101	1101	0000	001	RW	CONTEXTIDR_EL12	CONTI
11	101	1101	0000	111	RW	SCXTNUM_EL12	SCXTN
11	101	1110	0001	000	RW	CNTKCTL_EL12	CNTK(
11	101	1110	0010	000	RW	CNTP_TVAL_EL02	CNTP_
11	101	1110	0010	001	RW	CNTP_CTL_EL02	CNTP_
11	101	1110	0010	010	RW	CNTP_CVAL_EL02	CNTP_
11	101	1110	0011	000	RW	CNTV_TVAL_EL02	CNTV_
11	101	1110	0011	001	RW	CNTV_CTL_EL02	CNTV_
11	101	1110	0011	010	RW	CNTV_CVAL_EL02	CNTV_
11	110	0001	0000	000	RW	SCTLR_EL3	SCTLR
11	110	0001	0000	001	RW	ACTLR_EL3	ACTLI
11	110	0001	0000	011	RW	SCTLR2_EL3	SCTLR
11	110	0001	0001	000	RW	SCR_EL3	SCR_E
11	110	0001	0001	001	RW	SDER32_EL3	SDER3
11	110	0001	0001	010	RW	CPTR_EL3	CPTR_
11	110	0001	0001	101	RW	FGWTE3_EL3	FGWT
11	110	0001	0010	000	RW	ZCR_EL3	ZCR_E
11	110	0001	0010	110	RW	SMCR_EL3	SMCR_
11	110	0001	0011	001	RW	MDCR_EL3	MDCR
11	110	0010	0000	000	RW	TTBR0_EL3	TTBR0
11	110	0010	0000	010	RW	TCR_EL3	TCR_E
11	110	0010	0001	100	RW	GPTBR_EL3	GPTBI
11	110	0010	0001	101	RW	GPCBW_EL3	GPCBW
11	110	0010	0001	110	RW	GPCCR_EL3	GPCCI
11	110	0010	0101	000	RW	GCSCR_EL3	GCSCI
11	110	0010	0101	001	RW	GCSPR_EL3	GCSPI
11	110	0100	0000	000	RW	SPSR_EL3	SPSR_
11	110	0100	0000	001	RW	ELR_EL3	ELR_E
11	110	0100	0001	000	RW	SP_EL2	SP_EL
11	110	0101	0001	000	RW	AFSR0_EL3	AFSR0
11	110	0101	0001	001	RW	AFSR1_EL3	AFSR1
11	110	0101	0010	000	RW	ESR_EL3	ESR_E

op0	op1	CRn	CRm	op2	Access	Mnemonic	
11	110	0101	0010	011	RW	VSESR_EL3	VSESR
11	110	0101	0110	000	RW	TFSR_EL3	TFSR_
11	110	0110	0000	000	RW	FAR_EL3	FAR_E
11	110	0110	0000	101	RW	MFAR_EL3	MFAR_
11	110	1001	1010	011	RW	PMBSR_EL3	PMBSR
11	110	1001	1011	011	RW	TRBSR_EL3	TRBSR
11	110	1010	0001	001	RW	MAIR2_EL3	MAIR2
11	110	1010	0010	000	RW	MAIR_EL3	MAIR_
11	110	1010	0010	011	RW	PIR_EL3	PIR_EL
11	110	1010	0010	100	RW	POR_EL3	POR_E
11	110	1010	0011	000	RW	AMAIR_EL3	AMAIR
11	110	1010	0011	001	RW	AMAIR2_EL3	AMAIR
11	110	1010	0101	000	RW	MPAM3_EL3	MPAM
11	110	1010	0101	100	RW	MPAMBW3_EL3	MPAM
11	110	1010	1010	001	RW	MECID_RL_A_EL3	MECID
11	110	1100	0000	000	RW	VBAR_EL3	VBAR_
11	110	1100	0000	001	RO	RVBAR_EL3	RVBA
11	110	1100	0000	010	RW	RMR_EL3	RMR_
11	110	1100	0001	001	RW	VDISR_EL3	VDISR
11	110	1100	1100	100	RW	ICC_CTLR_EL3	ICC_C
11	110	1100	1100	101	RW	ICC_SRE_EL3	ICC_SR
11	110	1100	1100	111	RW	ICC_IGRPEN1_EL3	ICC_IG
11	110	1101	0000	010	RW	TPIDR_EL3	TPIDR
11	110	1101	0000	111	RW	SCXTNUM_EL3	SCXTN
11	111	1110	0010	000	RW	CNTPS_TVAL_EL1	CNTPS
11	111	1110	0010	001	RW	CNTPS_CTL_EL1	CNTPS
11	111	1110	0010	010	RW	CNTPS_CVAL_EL1	CNTPS
11	op1[2:0]	1x11	Cm[3:0]	op2[2:0]	RW	S3_<op1>_C<Cn>_C<Cm>_<op2>	S3_<op

Accessed using TLBI:

op0	op1	CRn	CRm	op2	Mnemonic
01	000	1000	0001	000	TLBI VMALLE1OS
01	000	1000	0001	001	TLBI VAE1OS
01	000	1000	0001	010	TLBI ASIDE1OS
01	000	1000	0001	011	TLBI VAAE1OS
01	000	1000	0001	101	TLBI VALE1OS
01	000	1000	0001	111	TLBI VAALE1OS
01	000	1000	0010	001	TLBI RVAE1IS
01	000	1000	0010	011	TLBI RVAAE1IS
01	000	1000	0010	101	TLBI RVALE1IS
01	000	1000	0010	111	TLBI RVAALE1IS
01	000	1000	0011	000	TLBI VMALLE1IS
01	000	1000	0011	001	TLBI VAE1IS
01	000	1000	0011	010	TLBI ASIDE1IS
01	000	1000	0011	011	TLBI VAAE1IS
01	000	1000	0011	101	TLBI VALE1IS
01	000	1000	0011	111	TLBI VAALE1IS
01	000	1000	0101	001	TLBI RVAE1OS

op0	op1	CRn	CRm	op2	Mnemonic
01	000	1000	0101	011	TLBI RVAAE1OS
01	000	1000	0101	101	TLBI RVALE1OS
01	000	1000	0101	111	TLBI RVAALE1OS
01	000	1000	0110	001	TLBI RVAE1
01	000	1000	0110	011	TLBI RVAAE1
01	000	1000	0110	101	TLBI RVALE1
01	000	1000	0110	111	TLBI RVAALE1
01	000	1000	0111	000	TLBI VMALLE1
01	000	1000	0111	001	TLBI VAE1
01	000	1000	0111	010	TLBI ASIDE1
01	000	1000	0111	011	TLBI VAAE1
01	000	1000	0111	101	TLBI VALE1
01	000	1000	0111	111	TLBI VAALE1
01	000	1001	0001	000	TLBI VMALLE1OSNXS
01	000	1001	0001	001	TLBI VAE1OSNXS
01	000	1001	0001	010	TLBI ASIDE1OSNXS
01	000	1001	0001	011	TLBI VAAE1OSNXS
01	000	1001	0001	101	TLBI VALE1OSNXS
01	000	1001	0001	111	TLBI VAALE1OSNXS
01	000	1001	0010	001	TLBI RVAE1ISNXS
01	000	1001	0010	011	TLBI RVAAE1ISNXS
01	000	1001	0010	101	TLBI RVALE1ISNXS
01	000	1001	0010	111	TLBI RVAALE1ISNXS
01	000	1001	0011	000	TLBI VMALLE1ISNXS
01	000	1001	0011	001	TLBI VAE1ISNXS
01	000	1001	0011	010	TLBI ASIDE1ISNXS
01	000	1001	0011	011	TLBI VAAE1ISNXS
01	000	1001	0011	101	TLBI VALE1ISNXS
01	000	1001	0011	111	TLBI VAALE1ISNXS
01	000	1001	0101	001	TLBI RVAE1OSNXS
01	000	1001	0101	011	TLBI RVAAE1OSNXS
01	000	1001	0101	101	TLBI RVALE1OSNXS
01	000	1001	0101	111	TLBI RVAALE1OSNXS
01	000	1001	0110	001	TLBI RVAE1NXS
01	000	1001	0110	011	TLBI RVAAE1NXS
01	000	1001	0110	101	TLBI RVALE1NXS
01	000	1001	0110	111	TLBI RVAALE1NXS
01	000	1001	0111	000	TLBI VMALLE1NXS
01	000	1001	0111	001	TLBI VAE1NXS
01	000	1001	0111	010	TLBI ASIDE1NXS
01	000	1001	0111	011	TLBI VAAE1NXS
01	000	1001	0111	101	TLBI VALE1NXS
01	000	1001	0111	111	TLBI VAALE1NXS
01	100	1000	0000	001	TLBI IPAS2E1IS
01	100	1000	0000	010	TLBI RIPAS2E1IS
01	100	1000	0000	101	TLBI IPAS2LE1IS
01	100	1000	0000	110	TLBI RIPAS2LE1IS
01	100	1000	0001	000	TLBI ALLE2OS
01	100	1000	0001	001	TLBI VAE2OS

op0	op1	CRn	CRm	op2	Mnemonic
01	100	1000	0001	100	TLBI ALLE1OS
01	100	1000	0001	101	TLBI VALE2OS
01	100	1000	0001	110	TLBI VMALLS12E1OS
01	100	1000	0010	001	TLBI RVAE2IS
01	100	1000	0010	010	TLBI VMALLWS2E1IS
01	100	1000	0010	101	TLBI RVALE2IS
01	100	1000	0011	000	TLBI ALLE2IS
01	100	1000	0011	001	TLBI VAE2IS
01	100	1000	0011	100	TLBI ALLE1IS
01	100	1000	0011	101	TLBI VALE2IS
01	100	1000	0011	110	TLBI VMALLS12E1IS
01	100	1000	0100	000	TLBI IPAS2E1OS
01	100	1000	0100	001	TLBI IPAS2E1
01	100	1000	0100	010	TLBI RIPAS2E1
01	100	1000	0100	011	TLBI RIPAS2E1OS
01	100	1000	0100	100	TLBI IPAS2LE1OS
01	100	1000	0100	101	TLBI IPAS2LE1
01	100	1000	0100	110	TLBI RIPAS2LE1
01	100	1000	0100	111	TLBI RIPAS2LE1OS
01	100	1000	0101	001	TLBI RVAE2OS
01	100	1000	0101	010	TLBI VMALLWS2E1OS
01	100	1000	0101	101	TLBI RVALE2OS
01	100	1000	0110	001	TLBI RVAE2
01	100	1000	0110	010	TLBI VMALLWS2E1
01	100	1000	0110	101	TLBI RVALE2
01	100	1000	0111	000	TLBI ALLE2
01	100	1000	0111	001	TLBI VAE2
01	100	1000	0111	100	TLBI ALLE1
01	100	1000	0111	101	TLBI VALE2
01	100	1000	0111	110	TLBI VMALLS12E1
01	100	1001	0000	001	TLBI IPAS2E1ISNXS
01	100	1001	0000	010	TLBI RIPAS2E1ISNXS
01	100	1001	0000	101	TLBI IPAS2LE1ISNXS
01	100	1001	0000	110	TLBI RIPAS2LE1ISNXS
01	100	1001	0001	000	TLBI ALLE2OSNXS
01	100	1001	0001	001	TLBI VAE2OSNXS
01	100	1001	0001	100	TLBI ALLE1OSNXS
01	100	1001	0001	101	TLBI VALE2OSNXS
01	100	1001	0001	110	TLBI VMALLS12E1OSNXS
01	100	1001	0010	001	TLBI RVAE2ISNXS
01	100	1001	0010	010	TLBI VMALLWS2E1ISNXS
01	100	1001	0010	101	TLBI RVALE2ISNXS
01	100	1001	0011	000	TLBI ALLE2ISNXS
01	100	1001	0011	001	TLBI VAE2ISNXS
01	100	1001	0011	100	TLBI ALLE1ISNXS
01	100	1001	0011	101	TLBI VALE2ISNXS
01	100	1001	0011	110	TLBI VMALLS12E1ISNXS
01	100	1001	0100	000	TLBI IPAS2E1OSNXS
01	100	1001	0100	001	TLBI IPAS2E1NXS

op0	op1	CRn	CRm	op2	Mnemonic
01	100	1001	0100	010	TLBI RIPAS2E1NXS
01	100	1001	0100	011	TLBI RIPAS2E1OSNXS
01	100	1001	0100	100	TLBI IPAS2LE1OSNXS
01	100	1001	0100	101	TLBI IPAS2LE1NXS
01	100	1001	0100	110	TLBI RIPAS2LE1NXS
01	100	1001	0100	111	TLBI RIPAS2LE1OSNXS
01	100	1001	0101	001	TLBI RVAE2OSNXS
01	100	1001	0101	010	TLBI VMALLWS2E1OSNXS
01	100	1001	0101	101	TLBI RVALE2OSNXS
01	100	1001	0110	001	TLBI RVAE2NXS
01	100	1001	0110	010	TLBI VMALLWS2E1NXS
01	100	1001	0110	101	TLBI RVALE2NXS
01	100	1001	0111	000	TLBI ALLE2NXS
01	100	1001	0111	001	TLBI VAE2NXS
01	100	1001	0111	100	TLBI ALLE1NXS
01	100	1001	0111	101	TLBI VALE2NXS
01	100	1001	0111	110	TLBI VMALLS12E1NXS
01	110	1000	0001	000	TLBI ALLE3OS
01	110	1000	0001	001	TLBI VAE3OS
01	110	1000	0001	100	TLBI PAALLOS
01	110	1000	0001	101	TLBI VALE3OS
01	110	1000	0010	001	TLBI RVAE3IS
01	110	1000	0010	101	TLBI RVALE3IS
01	110	1000	0011	000	TLBI ALLE3IS
01	110	1000	0011	001	TLBI VAE3IS
01	110	1000	0011	101	TLBI VALE3IS
01	110	1000	0100	011	TLBI RPAOS
01	110	1000	0100	111	TLBI RPALOS
01	110	1000	0101	001	TLBI RVAE3OS
01	110	1000	0101	101	TLBI RVALE3OS
01	110	1000	0110	001	TLBI RVAE3
01	110	1000	0110	101	TLBI RVALE3
01	110	1000	0111	000	TLBI ALLE3
01	110	1000	0111	001	TLBI VAE3
01	110	1000	0111	100	TLBI PAALL
01	110	1000	0111	101	TLBI VALE3
01	110	1001	0001	000	TLBI ALLE3OSNXS
01	110	1001	0001	001	TLBI VAE3OSNXS
01	110	1001	0001	101	TLBI VALE3OSNXS
01	110	1001	0010	001	TLBI RVAE3ISNXS
01	110	1001	0010	101	TLBI RVALE3ISNXS
01	110	1001	0011	000	TLBI ALLE3ISNXS
01	110	1001	0011	001	TLBI VAE3ISNXS
01	110	1001	0011	101	TLBI VALE3ISNXS
01	110	1001	0101	001	TLBI RVAE3OSNXS
01	110	1001	0101	101	TLBI RVALE3OSNXS
01	110	1001	0110	001	TLBI RVAE3NXS
01	110	1001	0110	101	TLBI RVALE3NXS
01	110	1001	0111	000	TLBI ALLE3NXS

op0	op1	CRn	CRm	op2	Mnemonic
01	110	1001	0111	001	TLBI VAE3NXS
01	110	1001	0111	101	TLBI VALE3NXS

Accessed using TLBIP:

op0	op1	CRn	CRm	op2	Mnemonic
01	000	1000	0001	001	TLBIP VAE1OS
01	000	1000	0001	011	TLBIP VAAE1OS
01	000	1000	0001	101	TLBIP VALE1OS
01	000	1000	0001	111	TLBIP VAALE1OS
01	000	1000	0010	001	TLBIP RVAE1IS
01	000	1000	0010	011	TLBIP RVAAE1IS
01	000	1000	0010	101	TLBIP RVALE1IS
01	000	1000	0010	111	TLBIP RVAALE1IS
01	000	1000	0011	001	TLBIP VAE1IS
01	000	1000	0011	011	TLBIP VAAE1IS
01	000	1000	0011	101	TLBIP VALE1IS
01	000	1000	0011	111	TLBIP VAALE1IS
01	000	1000	0101	001	TLBIP RVAE1OS
01	000	1000	0101	011	TLBIP RVAAE1OS
01	000	1000	0101	101	TLBIP RVALE1OS
01	000	1000	0101	111	TLBIP RVAALE1OS
01	000	1000	0110	001	TLBIP RVAE1
01	000	1000	0110	011	TLBIP RVAAE1
01	000	1000	0110	101	TLBIP RVALE1
01	000	1000	0110	111	TLBIP RVAALE1
01	000	1000	0111	001	TLBIP VAE1
01	000	1000	0111	011	TLBIP VAAE1
01	000	1000	0111	101	TLBIP VALE1
01	000	1000	0111	111	TLBIP VAALE1
01	000	1001	0001	001	TLBIP VAE1OSNXS
01	000	1001	0001	011	TLBIP VAAE1OSNXS
01	000	1001	0001	101	TLBIP VALE1OSNXS
01	000	1001	0001	111	TLBIP VAALE1OSNXS
01	000	1001	0010	001	TLBIP RVAE1ISNXS
01	000	1001	0010	011	TLBIP RVAAE1ISNXS
01	000	1001	0010	101	TLBIP RVALE1ISNXS
01	000	1001	0010	111	TLBIP RVAALE1ISNXS
01	000	1001	0011	001	TLBIP VAE1ISNXS
01	000	1001	0011	011	TLBIP VAAE1ISNXS
01	000	1001	0011	101	TLBIP VALE1ISNXS
01	000	1001	0011	111	TLBIP VAALE1ISNXS
01	000	1001	0101	001	TLBIP RVAE1OSNXS
01	000	1001	0101	011	TLBIP RVAAE1OSNXS
01	000	1001	0101	101	TLBIP RVALE1OSNXS
01	000	1001	0101	111	TLBIP RVAALE1OSNXS
01	000	1001	0110	001	TLBIP RVAE1NXS
01	000	1001	0110	011	TLBIP RVAAE1NXS
01	000	1001	0110	101	TLBIP RVALE1NXS

op0	op1	CRn	CRm	op2	Mnemonic
01	000	1001	0110	111	TLBIP RVAALE1NXS
01	000	1001	0111	001	TLBIP VAE1NXS
01	000	1001	0111	011	TLBIP VAAE1NXS
01	000	1001	0111	101	TLBIP VALE1NXS
01	000	1001	0111	111	TLBIP VAALE1NXS
01	100	1000	0000	001	TLBIP IPAS2E1IS
01	100	1000	0000	010	TLBIP RIPAS2E1IS
01	100	1000	0000	101	TLBIP IPAS2LE1IS
01	100	1000	0000	110	TLBIP RIPAS2LE1IS
01	100	1000	0001	001	TLBIP VAE2OS
01	100	1000	0001	101	TLBIP VALE2OS
01	100	1000	0010	001	TLBIP RVAE2IS
01	100	1000	0010	101	TLBIP RVALE2IS
01	100	1000	0011	001	TLBIP VAE2IS
01	100	1000	0011	101	TLBIP VALE2IS
01	100	1000	0100	000	TLBIP IPAS2E1OS
01	100	1000	0100	001	TLBIP IPAS2E1
01	100	1000	0100	010	TLBIP RIPAS2E1
01	100	1000	0100	011	TLBIP RIPAS2E1OS
01	100	1000	0100	100	TLBIP IPAS2LE1OS
01	100	1000	0100	101	TLBIP IPAS2LE1
01	100	1000	0100	110	TLBIP RIPAS2LE1
01	100	1000	0100	111	TLBIP RIPAS2LE1OS
01	100	1000	0101	001	TLBIP RVAE2OS
01	100	1000	0101	101	TLBIP RVALE2OS
01	100	1000	0110	001	TLBIP RVAE2
01	100	1000	0110	101	TLBIP RVALE2
01	100	1000	0111	001	TLBIP VAE2
01	100	1000	0111	101	TLBIP VALE2
01	100	1001	0000	001	TLBIP IPAS2E1ISNXS
01	100	1001	0000	010	TLBIP RIPAS2E1ISNXS
01	100	1001	0000	101	TLBIP IPAS2LE1ISNXS
01	100	1001	0000	110	TLBIP RIPAS2LE1ISNXS
01	100	1001	0001	001	TLBIP VAE2OSNXS
01	100	1001	0001	101	TLBIP VALE2OSNXS
01	100	1001	0010	001	TLBIP RVAE2ISNXS
01	100	1001	0010	101	TLBIP RVALE2ISNXS
01	100	1001	0011	001	TLBIP VAE2ISNXS
01	100	1001	0011	101	TLBIP VALE2ISNXS
01	100	1001	0100	000	TLBIP IPAS2E1OSNXS
01	100	1001	0100	001	TLBIP IPAS2E1NXS
01	100	1001	0100	010	TLBIP RIPAS2E1NXS
01	100	1001	0100	011	TLBIP RIPAS2E1OSNXS
01	100	1001	0100	100	TLBIP IPAS2LE1OSNXS
01	100	1001	0100	101	TLBIP IPAS2LE1NXS
01	100	1001	0100	110	TLBIP RIPAS2LE1NXS
01	100	1001	0100	111	TLBIP RIPAS2LE1OSNXS
01	100	1001	0101	001	TLBIP RVAE2OSNXS
01	100	1001	0101	101	TLBIP RVALE2OSNXS

op0	op1	CRn	CRm	op2	Mnemonic
01	100	1001	0110	001	TLBIP RVAE2NXS
01	100	1001	0110	101	TLBIP RVALE2NXS
01	100	1001	0111	001	TLBIP VAE2NXS
01	100	1001	0111	101	TLBIP VALE2NXS
01	110	1000	0001	001	TLBIP VAE3OS
01	110	1000	0001	101	TLBIP VALE3OS
01	110	1000	0010	001	TLBIP RVAE3IS
01	110	1000	0010	101	TLBIP RVALE3IS
01	110	1000	0011	001	TLBIP VAE3IS
01	110	1000	0011	101	TLBIP VALE3IS
01	110	1000	0101	001	TLBIP RVAE3OS
01	110	1000	0101	101	TLBIP RVALE3OS
01	110	1000	0110	001	TLBIP RVAE3
01	110	1000	0110	101	TLBIP RVALE3
01	110	1000	0111	001	TLBIP VAE3
01	110	1000	0111	101	TLBIP VALE3
01	110	1001	0001	001	TLBIP VAE3OSNXS
01	110	1001	0001	101	TLBIP VALE3OSNXS
01	110	1001	0010	001	TLBIP RVAE3ISNXS
01	110	1001	0010	101	TLBIP RVALE3ISNXS
01	110	1001	0011	001	TLBIP VAE3ISNXS
01	110	1001	0011	101	TLBIP VALE3ISNXS
01	110	1001	0101	001	TLBIP RVAE3OSNXS
01	110	1001	0101	101	TLBIP RVALE3OSNXS
01	110	1001	0110	001	TLBIP RVAE3NXS
01	110	1001	0110	101	TLBIP RVALE3NXS
01	110	1001	0111	001	TLBIP VAE3NXS
01	110	1001	0111	101	TLBIP VALE3NXS

Accessed using TRCIT:

op0	op1	CRn	CRm	op2	Mnemonic
01	011	0111	0010	111	TRCIT

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

Index by functional group

Below are indexes for registers with the following main functional groups:

In the IMP DEF functional group:

Exec state	Name	Description
AArch32	ACTLR	Auxiliary Control Register
AArch32	ACTLR2	Auxiliary Control Register 2
AArch64	ACTLR_EL1	Auxiliary Control Register (EL1)
AArch64	ACTLR_EL2	Auxiliary Control Register (EL2)
AArch64	ACTLR_EL3	Auxiliary Control Register (EL3)
AArch32	ADFSR	Auxiliary Data Fault Status Register
AArch64	AFSR0_EL1	Auxiliary Fault Status Register 0 (EL1)
AArch64	AFSR0_EL2	Auxiliary Fault Status Register 0 (EL2)
AArch64	AFSR0_EL3	Auxiliary Fault Status Register 0 (EL3)

Exec state	Name	Description
AArch64	AFSR1_EL1	Auxiliary Fault Status Register 1 (EL1)
AArch64	AFSR1_EL2	Auxiliary Fault Status Register 1 (EL2)
AArch64	AFSR1_EL3	Auxiliary Fault Status Register 1 (EL3)
AArch32	AIDR	Auxiliary ID Register
AArch64	AIDR_EL1	Auxiliary ID Register
AArch32	AIFSR	Auxiliary Instruction Fault Status Register
AArch32	AMAIR0	Auxiliary Memory Attribute Indirection Register 0
AArch32	AMAIR1	Auxiliary Memory Attribute Indirection Register 1
AArch64	AMAIR_EL1	Auxiliary Memory Attribute Indirection Register (EL1)
AArch64	AMAIR_EL2	Auxiliary Memory Attribute Indirection Register (EL2)
AArch64	AMAIR_EL3	Auxiliary Memory Attribute Indirection Register (EL3)
AArch64	HACR_EL2	Hypervisor Auxiliary Control Register
AArch32	HACTLR	Hyp Auxiliary Control Register
AArch32	HACTLR2	Hyp Auxiliary Control Register 2
AArch32	HADFSR	Hyp Auxiliary Data Fault Status Register
AArch32	HAIFSR	Hyp Auxiliary Instruction Fault Status Register
AArch32	HAMAIR0	Hyp Auxiliary Memory Attribute Indirection Register 0
AArch32	HAMAIR1	Hyp Auxiliary Memory Attribute Indirection Register 1
AArch64	S1_<op1>_<Cn>_<Cm>_<op2>	IMPLEMENTATION DEFINED System instructions
AArch64	S3_<op1>_<Cn>_<Cm>_<op2>	IMPLEMENTATION DEFINED Registers

In the Exception functional group:

Exec state	Name	Description
AArch32	ADFSR	Auxiliary Data Fault Status Register
AArch64	AFSR0_EL1	Auxiliary Fault Status Register 0 (EL1)
AArch64	AFSR0_EL2	Auxiliary Fault Status Register 0 (EL2)
AArch64	AFSR0_EL3	Auxiliary Fault Status Register 0 (EL3)
AArch64	AFSR1_EL1	Auxiliary Fault Status Register 1 (EL1)
AArch64	AFSR1_EL2	Auxiliary Fault Status Register 1 (EL2)
AArch64	AFSR1_EL3	Auxiliary Fault Status Register 1 (EL3)
AArch32	AIFSR	Auxiliary Instruction Fault Status Register
AArch32	DFAR	Data Fault Address Register
AArch32	DFSR	Data Fault Status Register
AArch64	ESR_EL1	Exception Syndrome Register (EL1)
AArch64	ESR_EL2	Exception Syndrome Register (EL2)
AArch64	ESR_EL3	Exception Syndrome Register (EL3)
AArch64	FAR_EL1	Fault Address Register (EL1)
AArch64	FAR_EL2	Fault Address Register (EL2)
AArch64	FAR_EL3	Fault Address Register (EL3)
AArch32	HADFSR	Hyp Auxiliary Data Fault Status Register
AArch32	HAIFSR	Hyp Auxiliary Instruction Fault Status Register
AArch32	HDFAR	Hyp Data Fault Address Register
AArch32	HIFAR	Hyp Instruction Fault Address Register
AArch32	HPFAR	Hyp IPA Fault Address Register
AArch64	HPFAR_EL2	Hypervisor IPA Fault Address Register
AArch32	HSR	Hyp Syndrome Register
AArch32	HVBAR	Hyp Vector Base Address Register
AArch32	IFAR	Instruction Fault Address Register
AArch32	IFSR	Instruction Fault Status Register
AArch64	IFSR32_EL2	Instruction Fault Status Register (EL2)
AArch32	ISR	Interrupt Status Register
AArch64	ISR_EL1	Interrupt Status Register
AArch32	MVBAR	Monitor Vector Base Address Register
AArch32	VBAR	Vector Base Address Register
AArch64	VBAR_EL1	Vector Base Address Register (EL1)
AArch64	VBAR_EL2	Vector Base Address Register (EL2)
AArch64	VBAR_EL3	Vector Base Address Register (EL3)

In the Memory functional group:

Exec state	Name	Description
AArch32	AMAIR0	Auxiliary Memory Attribute Indirection Register 0
AArch32	AMAIR1	Auxiliary Memory Attribute Indirection Register 1
AArch64	AMAIR2_EL1	Extended Auxiliary Memory Attribute Indirection Register (EL1)
AArch64	AMAIR2_EL2	Extended Auxiliary Memory Attribute Indirection Register (EL2)
AArch64	AMAIR2_EL3	Extended Auxiliary Memory Attribute Indirection Register (EL3)
AArch64	AMAIR_EL1	Auxiliary Memory Attribute Indirection Register (EL1)
AArch64	AMAIR_EL2	Auxiliary Memory Attribute Indirection Register (EL2)
AArch64	AMAIR_EL3	Auxiliary Memory Attribute Indirection Register (EL3)
AArch32	CONTEXTIDR	Context ID Register
AArch64	CONTEXTIDR_EL1	Context ID Register (EL1)
AArch64	CONTEXTIDR_EL2	Context ID Register (EL2)
AArch32	DACR	Domain Access Control Register
AArch64	DACR32_EL2	Domain Access Control Register
AArch64	GPCBW_EL3	Granule Protection Check Bypass Window Register (EL3)
AArch64	GPCCR_EL3	Granule Protection Check Control Register (EL3)
AArch64	GPTBR_EL3	Granule Protection Table Base Register
AArch64	HACDBSBR_EL2	Hardware Accelerator for Cleaning Dirty State Base Register
AArch64	HACDBSCONS_EL2	Hardware Accelerator for Cleaning Dirty State Consumer Register
AArch32	HAMAIR0	Hyp Auxiliary Memory Attribute Indirection Register 0
AArch32	HAMAIR1	Hyp Auxiliary Memory Attribute Indirection Register 1
AArch64	HDBSSBR_EL2	Hardware Dirty State Tracking Structure Base Register
AArch64	HDBSSPROD_EL2	Hardware Dirty State Tracking Structure Producer Register
AArch32	HMAIR0	Hyp Memory Attribute Indirection Register 0
AArch32	HMAIR1	Hyp Memory Attribute Indirection Register 1
AArch32	HTCR	Hyp Translation Control Register
AArch32	HTTBR	Hyp Translation Table Base Register
AArch64	LORC_EL1	LORegion Control (EL1)
AArch64	LOREA_EL1	LORegion End Address (EL1)
AArch64	LORID_EL1	LORegionID (EL1)
AArch64	LORN_EL1	LORegion Number (EL1)
AArch64	LORSA_EL1	LORegion Start Address (EL1)
AArch32	MAIR0	Memory Attribute Indirection Register 0
AArch32	MAIR1	Memory Attribute Indirection Register 1
AArch64	MAIR2_EL1	Extended Memory Attribute Indirection Register (EL1)
AArch64	MAIR2_EL2	Extended Memory Attribute Indirection Register (EL2)
AArch64	MAIR2_EL3	Extended Memory Attribute Indirection Register (EL3)
AArch64	MAIR_EL1	Memory Attribute Indirection Register (EL1)
AArch64	MAIR_EL2	Memory Attribute Indirection Register (EL2)
AArch64	MAIR_EL3	Memory Attribute Indirection Register (EL3)
AArch32	NMRR	Normal Memory Remap Register
AArch64	PIRE0_EL1	Permission Indirection Register 0 (EL1)
AArch64	PIRE0_EL2	Permission Indirection Register 0 (EL2)
AArch64	PIR_EL1	Permission Indirection Register 1 (EL1)
AArch64	PIR_EL2	Permission Indirection Register 2 (EL2)
AArch64	PIR_EL3	Permission Indirection Register 3 (EL3)
AArch64	POR_EL0	Permission Overlay Register 0 (EL0)
AArch64	POR_EL1	Permission Overlay Register 1 (EL1)
AArch64	POR_EL2	Permission Overlay Register 2 (EL2)
AArch64	POR_EL3	Permission Overlay Register 3 (EL3)
AArch32	PRRR	Primary Region Remap Register
AArch64	RCWMASK_EL1	Read Check Write Instruction Mask (EL1)
AArch64	RCWSMASK_EL1	Software Read Check Write Instruction Mask (EL1)
AArch64	S2PIR_EL2	Stage 2 Permission Indirection Register (EL2)
AArch64	S2POR_EL1	Stage 2 Permission Overlay Register (EL1)
AArch64	TCR2_EL1	Extended Translation Control Register (EL1)
AArch64	TCR2_EL2	Extended Translation Control Register (EL2)
AArch64	TCR_EL1	Translation Control Register (EL1)
AArch64	TCR_EL2	Translation Control Register (EL2)
AArch64	TCR_EL3	Translation Control Register (EL3)
AArch32	TTBCR	Translation Table Base Control Register
AArch32	TTBCR2	Translation Table Base Control Register 2
AArch32	TTBR0	Translation Table Base Register 0

Exec state	Name	Description
AArch64	TTBR0_EL1	Translation Table Base Register 0 (EL1)
AArch64	TTBR0_EL2	Translation Table Base Register 0 (EL2)
AArch64	TTBR0_EL3	Translation Table Base Register 0 (EL3)
AArch32	TTBR1	Translation Table Base Register 1
AArch64	TTBR1_EL1	Translation Table Base Register 1 (EL1)
AArch64	TTBR1_EL2	Translation Table Base Register 1 (EL2)
AArch32	VTCR	Virtualization Translation Control Register
AArch64	VTCR_EL2	Virtualization Translation Control Register
AArch32	VTTBR	Virtualization Translation Table Base Register
AArch64	VTTBR_EL2	Virtualization Translation Table Base Register

In the PSTATE functional group:

Exec state	Name	Description
AArch64	ALLINT	All Interrupt Mask Bit
AArch32	APSR	Application Program Status Register
AArch32	CPSR	Current Program Status Register
AArch64	CurrentEL	Current Exception Level
AArch64	DAIF	Interrupt Mask Bits
AArch64	DIT	Data Independent Timing
AArch64	NZCV	Condition Flags
AArch64	PAN	Privileged Access Never
AArch64	PM	Profiling Exception Mask
AArch64	SPSel	Stack Pointer Select
AArch64	SSBS	Speculative Store Bypass Safe
AArch64	SVCR	Streaming Vector Control Register
AArch64	TCO	Tag Check Override
AArch64	UAO	User Access Override

In the Address functional group:

Exec state	Name	Description
AArch64	AT S12E0R	Address Translate Stages 1 and 2 EL0 Read
AArch64	AT S12E0W	Address Translate Stages 1 and 2 EL0 Write
AArch64	AT S12E1R	Address Translate Stages 1 and 2 EL1 Read
AArch64	AT S12E1W	Address Translate Stages 1 and 2 EL1 Write
AArch64	AT S1E0R	Address Translate Stage 1 EL0 Read
AArch64	AT S1E0W	Address Translate Stage 1 EL0 Write
AArch64	AT S1E1A	Address Translate Stage 1 EL1 Without Permission checks
AArch64	AT S1E1R	Address Translate Stage 1 EL1 Read
AArch64	AT S1E1RP	Address Translate Stage 1 EL1 Read PAN
AArch64	AT S1E1W	Address Translate Stage 1 EL1 Write
AArch64	AT S1E1WP	Address Translate Stage 1 EL1 Write PAN
AArch64	AT S1E2A	Address Translate Stage 1 EL2 Without Permission checks
AArch64	AT S1E2R	Address Translate Stage 1 EL2 Read
AArch64	AT S1E2W	Address Translate Stage 1 EL2 Write
AArch64	AT S1E3A	Address Translate Stage 1 EL3 Without Permission checks
AArch64	AT S1E3R	Address Translate Stage 1 EL3 Read
AArch64	AT S1E3W	Address Translate Stage 1 EL3 Write
AArch32	ATS12NSOPR	Address Translate Stages 1 and 2 Non-secure Only PL1 Read
AArch32	ATS12NSOPW	Address Translate Stages 1 and 2 Non-secure Only PL1 Write
AArch32	ATS12NSOUR	Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read
AArch32	ATS12NSOUW	Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write
AArch32	ATS1CPR	Address Translate Stage 1 Current state PL1 Read
AArch32	ATS1CPRP	Address Translate Stage 1 Current state PL1 Read PAN
AArch32	ATS1CPW	Address Translate Stage 1 Current state PL1 Write
AArch32	ATS1CPWP	Address Translate Stage 1 Current state PL1 Write PAN
AArch32	ATS1CUR	Address Translate Stage 1 Current state Unprivileged Read
AArch32	ATS1CUW	Address Translate Stage 1 Current state Unprivileged Write
AArch32	ATS1HR	Address Translate Stage 1 Hyp mode Read
AArch32	ATS1HW	Address Translate Stage 1 Hyp mode Write

In the Virt functional group:

Exec state	Name	Description
AArch64	ACTLR_EL2	Auxiliary Control Register (EL2)
AArch64	AFSR0_EL2	Auxiliary Fault Status Register 0 (EL2)
AArch64	AFSR1_EL2	Auxiliary Fault Status Register 1 (EL2)
AArch64	AMAIR_EL2	Auxiliary Memory Attribute Indirection Register (EL2)
AArch32	ATS1HR	Address Translate Stage 1 Hyp mode Read
AArch32	ATS1HW	Address Translate Stage 1 Hyp mode Write
AArch32	CNTHCTL	Counter-timer Hyp Control register
AArch64	CNTHCTL_EL2	Counter-timer Hypervisor Control Register
AArch64	CNTHPS_CTL_EL2	Counter-timer Secure Physical Timer Control Register (EL2)
AArch64	CNTHPS_CVAL_EL2	Counter-timer Secure Physical Timer CompareValue Register (EL2)
AArch64	CNTHPS_TVAL_EL2	Counter-timer Secure Physical Timer TimerValue Register (EL2)
AArch64	CNTHP_CTL_EL2	Counter-timer Hypervisor Physical Timer Control Register
AArch32	CNTHP_CVAL	Counter-timer Hyp Physical CompareValue register
AArch64	CNTHP_CVAL_EL2	Counter-timer Physical Timer CompareValue Register (EL2)
AArch32	CNTHP_TVAL	Counter-timer Hyp Physical Timer TimerValue register
AArch64	CNTHP_TVAL_EL2	Counter-timer Physical Timer TimerValue Register (EL2)
AArch32	CNTVOFF	Counter-timer Virtual Offset register
AArch64	CNTVOFF_EL2	Counter-timer Virtual Offset Register
AArch64	CPTR_EL2	Architectural Feature Trap Register (EL2)
AArch64	ESR_EL2	Exception Syndrome Register (EL2)
AArch64	FAR_EL2	Fault Address Register (EL2)
AArch64	HACDBSBR_EL2	Hardware Accelerator for Cleaning Dirty State Base Register
AArch64	HACDBSCONS_EL2	Hardware Accelerator for Cleaning Dirty State Consumer Register
AArch32	HACR	Hyp Auxiliary Configuration Register
AArch64	HACR_EL2	Hypervisor Auxiliary Control Register
AArch32	HACTLR	Hyp Auxiliary Control Register
AArch32	HACTLR2	Hyp Auxiliary Control Register 2
AArch32	HADFSR	Hyp Auxiliary Data Fault Status Register
AArch32	HAIFSR	Hyp Auxiliary Instruction Fault Status Register
AArch32	HAMAIR0	Hyp Auxiliary Memory Attribute Indirection Register 0
AArch32	HAMAIR1	Hyp Auxiliary Memory Attribute Indirection Register 1
AArch32	HCPTR	Hyp Architectural Feature Trap Register
AArch32	HCR	Hyp Configuration Register
AArch32	HCR2	Hyp Configuration Register 2
AArch64	HCRX_EL2	Extended Hypervisor Configuration Register
AArch64	HCR_EL2	Hypervisor Configuration Register
AArch64	HDBSSBR_EL2	Hardware Dirty State Tracking Structure Base Register
AArch64	HDBSSPROD_EL2	Hardware Dirty State Tracking Structure Producer Register
AArch32	HDCR	Hyp Debug Control Register
AArch32	HDFAR	Hyp Data Fault Address Register
AArch64	HFGITR2_EL2	Hypervisor Fine-Grained Instruction Trap Register 2
AArch32	HIFAR	Hyp Instruction Fault Address Register
AArch32	HMAIR0	Hyp Memory Attribute Indirection Register 0
AArch32	HMAIR1	Hyp Memory Attribute Indirection Register 1
AArch32	HPFAR	Hyp IPA Fault Address Register
AArch64	HPFAR_EL2	Hypervisor IPA Fault Address Register
AArch32	HRMR	Hyp Reset Management Register
AArch32	HSCTLR	Hyp System Control Register
AArch32	HSR	Hyp Syndrome Register
AArch32	HSTR	Hyp System Trap Register
AArch64	HSTR_EL2	Hypervisor System Trap Register
AArch32	HTCR	Hyp Translation Control Register
AArch32	HTPIDR	Hyp Software Thread ID Register
AArch32	HTRFCR	Hyp Trace Filter Control Register
AArch32	HTTBR	Hyp Translation Table Base Register
AArch32	HVBAR	Hyp Vector Base Address Register
AArch32	ICC_HSRE	Interrupt Controller Hyp System Register Enable register
AArch64	ICC_SRE_EL2	Interrupt Controller System Register Enable Register (EL2)
AArch32	ICH_AP0R<n>	Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch64	ICH_AP0R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch32	ICH_AP1R<n>	Interrupt Controller Hyp Active Priorities Group 1 Registers
AArch64	ICH_AP1R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 1 Registers

Exec state	Name	Description
AArch32	ICH_EISR	Interrupt Controller End of Interrupt Status Register
AArch64	ICH_EISR_EL2	Interrupt Controller End of Interrupt Status Register
AArch32	ICH_ELRSR	Interrupt Controller Empty List Register Status Register
AArch64	ICH_ELRSR_EL2	Interrupt Controller Empty List Register Status Register
AArch32	ICH_HCR	Interrupt Controller Hyp Control Register
AArch64	ICH_HCR_EL2	Interrupt Controller Hyp Control Register
AArch32	ICH_LR<n>	Interrupt Controller List Registers
AArch64	ICH_LR<n>_EL2	Interrupt Controller List Registers
AArch32	ICH_LRC<n>	Interrupt Controller List Registers
AArch32	ICH_MISR	Interrupt Controller Maintenance Interrupt State Register
AArch64	ICH_MISR_EL2	Interrupt Controller Maintenance Interrupt State Register
AArch32	ICH_VMCR	Interrupt Controller Virtual Machine Control Register
AArch64	ICH_VMCR_EL2	Interrupt Controller Virtual Machine Control Register
AArch32	ICH_VTR	Interrupt Controller VGIC Type Register
AArch64	ICH_VTR_EL2	Interrupt Controller VGIC Type Register
AArch64	MAIR_EL2	Memory Attribute Indirection Register (EL2)
AArch64	MDCR_EL2	Monitor Debug Configuration Register (EL2)
AArch64	RMR_EL2	Reset Management Register (EL2)
AArch64	SCTLR2_EL2	System Control Register (EL2)
AArch64	SCTLR_EL2	System Control Register (EL2)
AArch64	TCR2_EL2	Extended Translation Control Register (EL2)
AArch64	TCR_EL2	Translation Control Register (EL2)
AArch64	TLBI IPAS2E1	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBI IPAS2E1IS	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBI IPAS2E1OS	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBI IPAS2LE1	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBI IPAS2LE1IS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBI IPAS2LE1OS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TLBI RIPAS2E1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBI RIPAS2E1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBI RIPAS2E1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBI RIPAS2LE1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBI RIPAS2LE1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBI RIPAS2LE1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch32	TLBIALLH	TLB Invalidate All, Hyp mode
AArch32	TLBIALLHIS	TLB Invalidate All, Hyp mode, Inner Shareable
AArch32	TLBIIPAS2	TLB Invalidate by Intermediate Physical Address, Stage 2
AArch32	TLBIIPAS2IS	TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable
AArch32	TLBIIPAS2L	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level
AArch32	TLBIIPAS2LIS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable
AArch32	TLBIMVAH	TLB Invalidate by VA, Hyp mode
AArch32	TLBIMVAHIS	TLB Invalidate by VA, Hyp mode, Inner Shareable
AArch32	TLBIMVALH	TLB Invalidate by VA, Last level, Hyp mode
AArch32	TLBIMVALHIS	TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable
AArch64	TLBIP IPAS2E1	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBIP IPAS2E1IS	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBIP IPAS2E1OS	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBIP IPAS2LE1	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBIP IPAS2LE1IS	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBIP IPAS2LE1OS	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TLBIP RIPAS2E1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBIP RIPAS2E1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBIP RIPAS2E1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBIP RIPAS2LE1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBIP RIPAS2LE1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBIP RIPAS2LE1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TPIDR_EL2	EL2 Software Thread ID Register
AArch64	TTBR0_EL2	Translation Table Base Register 0 (EL2)
AArch64	TTBR1_EL2	Translation Table Base Register 1 (EL2)
AArch64	VBAR_EL2	Vector Base Address Register (EL2)
AArch32	VMPIDR	Virtualization Multiprocessor ID Register
AArch64	VMPIDR_EL2	Virtualization Multiprocessor ID Register
AArch32	VPIDR	Virtualization Processor ID Register
AArch64	VPIDR_EL2	Virtualization Processor ID Register

Exec state	Name	Description
AArch32	VTCR	Virtualization Translation Control Register
AArch64	VTCR_EL2	Virtualization Translation Control Register
AArch32	VTTBR	Virtualization Translation Table Base Register
AArch64	VTTBR_EL2	Virtualization Translation Table Base Register

In the Cache functional group:

Exec state	Name	Description
AArch32	BPIALL	Branch Predictor Invalidate All
AArch32	BPIALLIS	Branch Predictor Invalidate All, Inner Shareable
AArch32	BPIMVA	Branch Predictor Invalidate by VA
AArch64	DC CGDSW	Clean of Data and Allocation Tags by Set/Way
AArch64	DC CGDVAC	Clean of Data and Allocation Tags by VA to PoC
AArch64	DC CGDVADP	Clean of Data and Allocation Tags by VA to PoDP
AArch64	DC CGDVAOC	Clean of Data and Allocation Tags by VA to Outer Cache level
AArch64	DC CGDVAP	Clean of Data and Allocation Tags by VA to PoP
AArch64	DC CGSW	Clean of Allocation Tags by Set/Way
AArch64	DC CGVAC	Clean of Allocation Tags by VA to PoC
AArch64	DC CGVADP	Clean of Allocation Tags by VA to PoDP
AArch64	DC CGVAP	Clean of Allocation Tags by VA to PoP
AArch64	DC CIGDPAE	Clean and invalidate of data and allocation tags by PA to PoE
AArch64	DC CIGDPAPA	Clean and Invalidate of Data and Allocation Tags by PA to PoPA
AArch64	DC CIGDSW	Clean and Invalidate of Data and Allocation Tags by Set/Way
AArch64	DC CIGDVAC	Clean and Invalidate of Data and Allocation Tags by VA to PoC
AArch64	DC CIGDVAOC	Clean and Invalidate of Data and Allocation Tags by VA to Outer Cache level
AArch64	DC CIGDVAPS	Clean and Invalidate of Data and Allocation Tags by VA to PoPS
AArch64	DC CIGSW	Clean and Invalidate of Allocation Tags by Set/Way
AArch64	DC CIGVAC	Clean and Invalidate of Allocation Tags by VA to PoC
AArch64	DC CIPAE	Data or unified Cache line Clean and Invalidate by PA to PoE
AArch64	DC CIPAPA	Data or unified Cache line Clean and Invalidate by PA to PoPA
AArch64	DC CISW	Data or unified Cache line Clean and Invalidate by Set/Way
AArch64	DC CIVAC	Data or unified Cache line Clean and Invalidate by VA to PoC
AArch64	DC CIVAOC	Data or unified Cache line Clean and Invalidate by VA to Outer Cache level
AArch64	DC CIVAPS	Clean and Invalidate of Data by VA to PoPS
AArch64	DC CSW	Data or unified Cache line Clean by Set/Way
AArch64	DC CVAC	Data or unified Cache line Clean by VA to PoC
AArch64	DC CVADP	Data or unified Cache line Clean by VA to PoDP
AArch64	DC CVAOC	Data or unified Cache line Clean by VA to Outer Cache level
AArch64	DC CVAP	Data or unified Cache line Clean by VA to PoP
AArch64	DC CVAU	Data or unified Cache line Clean by VA to PoU
AArch64	DC GVA	Data Cache set Allocation Tag by VA
AArch64	DC GZVA	Data Cache set Allocation Tags and Zero by VA
AArch64	DC IGDSW	Invalidate of Data and Allocation Tags by Set/Way
AArch64	DC IGDVAC	Invalidate of Data and Allocation Tags by VA to PoC
AArch64	DC IGSW	Invalidate of Allocation Tags by Set/Way
AArch64	DC IGVAC	Invalidate of Allocation Tags by VA to PoC
AArch64	DC ISW	Data or unified Cache line Invalidate by Set/Way
AArch64	DC IVAC	Data or unified Cache line Invalidate by VA to PoC
AArch64	DC ZVA	Data Cache Zero by VA
AArch32	DCCIMVAC	Data Cache line Clean and Invalidate by VA to PoC
AArch32	DCCISW	Data Cache line Clean and Invalidate by Set/Way
AArch32	DCCMVAC	Data Cache line Clean by VA to PoC
AArch32	DCCMVAU	Data Cache line Clean by VA to PoU
AArch32	DCCSW	Data Cache line Clean by Set/Way
AArch32	DCIMVAC	Data Cache line Invalidate by VA to PoC
AArch32	DCISW	Data Cache line Invalidate by Set/Way
AArch64	IC IALLU	Instruction Cache Invalidate All to PoU
AArch64	IC IALLUIS	Instruction Cache Invalidate All to PoU, Inner Shareable
AArch64	IC IVAU	Instruction Cache line Invalidate by VA to PoU
AArch32	ICIALLU	Instruction Cache Invalidate All to PoU
AArch32	ICIALLUIS	Instruction Cache Invalidate All to PoU, Inner Shareable
AArch32	ICIMVAU	Instruction Cache line Invalidate by VA to PoU

In the Identification Registers functional group:

Exec state	Name	Description
AArch32	CCSIDR	Current Cache Size ID Register
AArch32	CCSIDR2	Current Cache Size ID Register 2
AArch64	CCSIDR2_EL1	Current Cache Size ID Register 2
AArch64	CCSIDR_EL1	Current Cache Size ID Register
AArch32	CLIDR	Cache Level ID Register
AArch64	CLIDR_EL1	Cache Level ID Register
AArch32	CSSELR	Cache Size Selection Register
AArch64	CSSELR_EL1	Cache Size Selection Register
AArch32	CTR	Cache Type Register
AArch64	CTR_EL0	Cache Type Register
AArch64	DCZID_EL0	Data Cache Zero ID Register
External	EDAA32PFR	External Debug Auxiliary Processor Feature Register
External	EDDFR	External Debug Feature Register
External	EDDFR1	External Debug Feature Register 1
External	EDDFR2	External Debug Feature Register 2
External	EDPFR	External Debug Processor Feature Register
AArch64	GMID_EL1	Multiple tag transfer ID Register
AArch64	ID_AA64AFR0_EL1	AArch64 Auxiliary Feature Register 0
AArch64	ID_AA64AFR1_EL1	AArch64 Auxiliary Feature Register 1
AArch64	ID_AA64DFR0_EL1	AArch64 Debug Feature Register 0
AArch64	ID_AA64DFR1_EL1	AArch64 Debug Feature Register 1
AArch64	ID_AA64DFR2_EL1	AArch64 Debug Feature Register 2
AArch64	ID_AA64FPFR0_EL1	AArch64 Floating-point Feature Register 0
AArch64	ID_AA64ISAR0_EL1	AArch64 Instruction Set Attribute Register 0
AArch64	ID_AA64ISAR1_EL1	AArch64 Instruction Set Attribute Register 1
AArch64	ID_AA64ISAR2_EL1	AArch64 Instruction Set Attribute Register 2
AArch64	ID_AA64ISAR3_EL1	AArch64 Instruction Set Attribute Register 3
AArch64	ID_AA64MMFR0_EL1	AArch64 Memory Model Feature Register 0
AArch64	ID_AA64MMFR1_EL1	AArch64 Memory Model Feature Register 1
AArch64	ID_AA64MMFR2_EL1	AArch64 Memory Model Feature Register 2
AArch64	ID_AA64MMFR3_EL1	AArch64 Memory Model Feature Register 3
AArch64	ID_AA64MMFR4_EL1	AArch64 Memory Model Feature Register 4
AArch64	ID_AA64PFR0_EL1	AArch64 Processor Feature Register 0
AArch64	ID_AA64PFR1_EL1	AArch64 Processor Feature Register 1
AArch64	ID_AA64PFR2_EL1	AArch64 Processor Feature Register 2
AArch64	ID_AA64SMFR0_EL1	SME Feature ID Register 0
AArch64	ID_AA64ZFR0_EL1	SVE Feature ID Register 0
AArch32	ID_AFR0	Auxiliary Feature Register 0
AArch64	ID_AFR0_EL1	AArch32 Auxiliary Feature Register 0
AArch32	ID_DFR0	Debug Feature Register 0
AArch64	ID_DFR0_EL1	AArch32 Debug Feature Register 0
AArch32	ID_DFR1	Debug Feature Register 1
AArch64	ID_DFR1_EL1	AArch32 Debug Feature Register 1
AArch32	ID_ISAR0	Instruction Set Attribute Register 0
AArch64	ID_ISAR0_EL1	AArch32 Instruction Set Attribute Register 0
AArch32	ID_ISAR1	Instruction Set Attribute Register 1
AArch64	ID_ISAR1_EL1	AArch32 Instruction Set Attribute Register 1
AArch32	ID_ISAR2	Instruction Set Attribute Register 2
AArch64	ID_ISAR2_EL1	AArch32 Instruction Set Attribute Register 2
AArch32	ID_ISAR3	Instruction Set Attribute Register 3
AArch64	ID_ISAR3_EL1	AArch32 Instruction Set Attribute Register 3
AArch32	ID_ISAR4	Instruction Set Attribute Register 4
AArch64	ID_ISAR4_EL1	AArch32 Instruction Set Attribute Register 4
AArch32	ID_ISAR5	Instruction Set Attribute Register 5
AArch64	ID_ISAR5_EL1	AArch32 Instruction Set Attribute Register 5
AArch32	ID_ISAR6	Instruction Set Attribute Register 6
AArch64	ID_ISAR6_EL1	AArch32 Instruction Set Attribute Register 6
AArch32	ID_MMFR0	Memory Model Feature Register 0
AArch64	ID_MMFR0_EL1	AArch32 Memory Model Feature Register 0
AArch32	ID_MMFR1	Memory Model Feature Register 1
AArch64	ID_MMFR1_EL1	AArch32 Memory Model Feature Register 1
AArch32	ID_MMFR2	Memory Model Feature Register 2

Exec state	Name	Description
AArch64	ID_MMFR2_EL1	AArch32 Memory Model Feature Register 2
AArch32	ID_MMFR3	Memory Model Feature Register 3
AArch64	ID_MMFR3_EL1	AArch32 Memory Model Feature Register 3
AArch32	ID_MMFR4	Memory Model Feature Register 4
AArch64	ID_MMFR4_EL1	AArch32 Memory Model Feature Register 4
AArch32	ID_MMFR5	Memory Model Feature Register 5
AArch64	ID_MMFR5_EL1	AArch32 Memory Model Feature Register 5
AArch32	ID_PFR0	Processor Feature Register 0
AArch64	ID_PFR0_EL1	AArch32 Processor Feature Register 0
AArch32	ID_PFR1	Processor Feature Register 1
AArch64	ID_PFR1_EL1	AArch32 Processor Feature Register 1
AArch32	ID_PFR2	Processor Feature Register 2
AArch64	ID_PFR2_EL1	AArch32 Processor Feature Register 2
AArch32	MIDR	Main ID Register
AArch64	MIDR_EL1	Main ID Register
External	MIDR_EL1	Main ID Register
AArch64	MPAMIDR_EL1	MPAM ID Register (EL1)
AArch32	MPIDR	Multiprocessor Affinity Register
AArch64	MPIDR_EL1	Multiprocessor Affinity Register
AArch32	REVIDR	Revision ID Register
AArch64	REVIDR_EL1	Revision ID Register
AArch64	SMIDR_EL1	Streaming Mode Identification Register
AArch32	TCMTR	TCM Type Register
AArch32	TLBTR	TLB Type Register

In the Predictor Maintenance Instructions functional group:

Exec state	Name	Description
AArch64	CFP RCTX	Control Flow Prediction Restriction by Context
AArch32	CFPRCTX	Control Flow Prediction Restriction by Context
AArch64	COSP RCTX	Clear Other Speculative Prediction Restriction by Context
AArch64	CPP RCTX	Cache Prefetch Prediction Restriction by Context
AArch32	CPPRCTX	Cache Prefetch Prediction Restriction by Context
AArch64	DVP RCTX	Data Value Prediction Restriction by Context
AArch32	DVPRCTX	Data Value Prediction Restriction by Context

In the Timer functional group:

Exec state	Name	Description
External	CNTACR<n>	Counter-timer Access Control Registers
External	CNTCR	Counter Control Register
External	CNTCV	Counter Count Value register
External	CNTEL0ACR	Counter-timer EL0 Access Control Register
External	CNTFID0	Counter Frequency ID
External	CNTFID<n>	Counter Frequency IDs, n > 0
AArch32	CNTFRQ	Counter-timer Frequency register
External	CNTFRQ	Counter-timer Frequency
AArch64	CNTFRQ_EL0	Counter-timer Frequency Register
AArch32	CNTHPS_CTL	Counter-timer Secure Physical Timer Control Register (EL2)
AArch32	CNTHPS_CVAL	Counter-timer Secure Physical Timer CompareValue Register (EL2)
AArch32	CNTHPS_TVAL	Counter-timer Secure Physical Timer TimerValue Register (EL2)
AArch32	CNTHP_CTL	Counter-timer Hyp Physical Timer Control register
AArch32	CNTHVS_CTL	Counter-timer Secure Virtual Timer Control Register (EL2)
AArch64	CNTHVS_CTL_EL2	Counter-timer Secure Virtual Timer Control Register (EL2)
AArch32	CNTHVS_CVAL	Counter-timer Secure Virtual Timer CompareValue Register (EL2)
AArch64	CNTHVS_CVAL_EL2	Counter-timer Secure Virtual Timer CompareValue Register (EL2)
AArch32	CNTHVS_TVAL	Counter-timer Secure Virtual Timer TimerValue Register (EL2)
AArch64	CNTHVS_TVAL_EL2	Counter-timer Secure Virtual Timer TimerValue Register (EL2)
AArch32	CNTHV_CTL	Counter-timer Virtual Timer Control register (EL2)
AArch64	CNTHV_CTL_EL2	Counter-timer Virtual Timer Control Register (EL2)
AArch32	CNTHV_CVAL	Counter-timer Virtual Timer CompareValue register (EL2)
AArch64	CNTHV_CVAL_EL2	Counter-timer Virtual Timer CompareValue Register (EL2)

Exec state	Name	Description
AArch32	CNTHV_TVAL	Counter-timer Virtual Timer TimerValue register (EL2)
AArch64	CNTHV_TVAL_EL2	Counter-timer Virtual Timer TimerValue Register (EL2)
External	CNTID	Counter Identification Register
AArch32	CNTKCTL	Counter-timer Kernel Control register
AArch64	CNTKCTL_EL1	Counter-timer Kernel Control Register
External	CNTNSAR	Counter-timer Non-secure Access Register
AArch32	CNTPCT	Counter-timer Physical Count register
External	CNTPCT	Counter-timer Physical Count
AArch32	CNTPCTSS	Counter-timer Self-Synchronized Physical Count register
AArch64	CNTPCTSS_EL0	Counter-timer Self-Synchronized Physical Count Register
AArch64	CNTPCT_EL0	Counter-timer Physical Count Register
AArch64	CNTPOFF_EL2	Counter-timer Physical Offset Register
AArch64	CNTPS_CTL_EL1	Counter-timer Physical Secure Timer Control Register
AArch64	CNTPS_CVAL_EL1	Counter-timer Physical Secure Timer CompareValue Register
AArch64	CNTPS_TVAL_EL1	Counter-timer Physical Secure Timer TimerValue Register
AArch32	CNTP_CTL	Counter-timer Physical Timer Control register
External	CNTP_CTL	Counter-timer Physical Timer Control
AArch64	CNTP_CTL_EL0	Counter-timer Physical Timer Control Register
AArch32	CNTP_CVAL	Counter-timer Physical Timer CompareValue register
External	CNTP_CVAL	Counter-timer Physical Timer CompareValue
AArch64	CNTP_CVAL_EL0	Counter-timer Physical Timer CompareValue Register
AArch32	CNTP_TVAL	Counter-timer Physical Timer TimerValue register
External	CNTP_TVAL	Counter-timer Physical Timer TimerValue
AArch64	CNTP_TVAL_EL0	Counter-timer Physical Timer TimerValue Register
External	CNTSCR	Counter Scale Register
External	CNTSR	Counter Status Register
External	CNTTIDR	Counter-timer Timer ID Register
AArch32	CNTVCT	Counter-timer Virtual Count register
External	CNTVCT	Counter-timer Virtual Count
AArch32	CNTVCTSS	Counter-timer Self-Synchronized Virtual Count register
AArch64	CNTVCTSS_EL0	Counter-timer Self-Synchronized Virtual Count Register
AArch64	CNTVCT_EL0	Counter-timer Virtual Count Register
External	CNTVOFF	Counter-timer Virtual Offset
External	CNTVOFF<n>	Counter-timer Virtual Offsets
AArch32	CNTV_CTL	Counter-timer Virtual Timer Control register
External	CNTV_CTL	Counter-timer Virtual Timer Control
AArch64	CNTV_CTL_EL0	Counter-timer Virtual Timer Control Register
AArch32	CNTV_CVAL	Counter-timer Virtual Timer CompareValue register
External	CNTV_CVAL	Counter-timer Virtual Timer CompareValue
AArch64	CNTV_CVAL_EL0	Counter-timer Virtual Timer CompareValue Register
AArch32	CNTV_TVAL	Counter-timer Virtual Timer TimerValue register
External	CNTV_TVAL	Counter-timer Virtual Timer TimerValue
AArch64	CNTV_TVAL_EL0	Counter-timer Virtual Timer TimerValue Register
External	CounterID<n>	Counter ID registers

In the TLB functional group:

Exec state	Name	Description
AArch32	COSPRCTX	Clear Other Speculative Prediction Restriction by Context
AArch32	DTLBIALL	Data TLB Invalidate All
AArch32	DTLBIASID	Data TLB Invalidate by ASID match
AArch32	DTLBIMVA	Data TLB Invalidate by VA
AArch32	ITLBIALL	Instruction TLB Invalidate All
AArch32	ITLBIASID	Instruction TLB Invalidate by ASID match
AArch32	ITLBIMVA	Instruction TLB Invalidate by VA
AArch64	TLBI ALLE1	TLB Invalidate All, EL1
AArch64	TLBI ALLE1IS	TLB Invalidate All, EL1, Inner Shareable
AArch64	TLBI ALLE1OS	TLB Invalidate All, EL1, Outer Shareable
AArch64	TLBI ALLE2	TLB Invalidate All, EL2
AArch64	TLBI ALLE2IS	TLB Invalidate All, EL2, Inner Shareable
AArch64	TLBI ALLE2OS	TLB Invalidate All, EL2, Outer Shareable
AArch64	TLBI ALLE3	TLB Invalidate All, EL3
AArch64	TLBI ALLE3IS	TLB Invalidate All, EL3, Inner Shareable

Exec state	Name	Description
AArch64	TLBI ALLE3OS	TLB Invalidate All, EL3, Outer Shareable
AArch64	TLBI ASIDE1	TLB Invalidate by ASID, EL1
AArch64	TLBI ASIDE1IS	TLB Invalidate by ASID, EL1, Inner Shareable
AArch64	TLBI ASIDE1OS	TLB Invalidate by ASID, EL1, Outer Shareable
AArch64	TLBI IPAS2E1	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBI IPAS2E1IS	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBI IPAS2E1OS	TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBI IPAS2LE1	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBI IPAS2LE1IS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBI IPAS2LE1OS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TLBI PAALL	TLB Invalidate GPT Information by PA, All Entries, Local
AArch64	TLBI PAALLOS	TLB Invalidate GPT Information by PA, All Entries, Outer Shareable
AArch64	TLBI RIPAS2E1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBI RIPAS2E1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBI RIPAS2E1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBI RIPAS2LE1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBI RIPAS2LE1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBI RIPAS2LE1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TLBI RPALOS	TLB Range Invalidate GPT Information by PA, Last level, Outer Shareable
AArch64	TLBI RPAOS	TLB Range Invalidate GPT Information by PA, Outer Shareable
AArch64	TLBI RVAAE1	TLB Range Invalidate by VA, All ASID, EL1
AArch64	TLBI RVAAE1IS	TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable
AArch64	TLBI RVAAE1OS	TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable
AArch64	TLBI RVAALE1	TLB Range Invalidate by VA, All ASID, Last level, EL1
AArch64	TLBI RVAALE1IS	TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable
AArch64	TLBI RVAALE1OS	TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable
AArch64	TLBI RVAE1	TLB Range Invalidate by VA, EL1
AArch64	TLBI RVAE1IS	TLB Range Invalidate by VA, EL1, Inner Shareable
AArch64	TLBI RVAE1OS	TLB Range Invalidate by VA, EL1, Outer Shareable
AArch64	TLBI RVAE2	TLB Range Invalidate by VA, EL2
AArch64	TLBI RVAE2IS	TLB Range Invalidate by VA, EL2, Inner Shareable
AArch64	TLBI RVAE2OS	TLB Range Invalidate by VA, EL2, Outer Shareable
AArch64	TLBI RVAE3	TLB Range Invalidate by VA, EL3
AArch64	TLBI RVAE3IS	TLB Range Invalidate by VA, EL3, Inner Shareable
AArch64	TLBI RVAE3OS	TLB Range Invalidate by VA, EL3, Outer Shareable
AArch64	TLBI RVALE1	TLB Range Invalidate by VA, Last level, EL1
AArch64	TLBI RVALE1IS	TLB Range Invalidate by VA, Last level, EL1, Inner Shareable
AArch64	TLBI RVALE1OS	TLB Range Invalidate by VA, Last level, EL1, Outer Shareable
AArch64	TLBI RVALE2	TLB Range Invalidate by VA, Last level, EL2
AArch64	TLBI RVALE2IS	TLB Range Invalidate by VA, Last level, EL2, Inner Shareable
AArch64	TLBI RVALE2OS	TLB Range Invalidate by VA, Last level, EL2, Outer Shareable
AArch64	TLBI RVALE3	TLB Range Invalidate by VA, Last level, EL3
AArch64	TLBI RVALE3IS	TLB Range Invalidate by VA, Last level, EL3, Inner Shareable
AArch64	TLBI RVALE3OS	TLB Range Invalidate by VA, Last level, EL3, Outer Shareable
AArch64	TLBI VAAE1	TLB Invalidate by VA, All ASID, EL1
AArch64	TLBI VAAE1IS	TLB Invalidate by VA, All ASID, EL1, Inner Shareable
AArch64	TLBI VAAE1OS	TLB Invalidate by VA, All ASID, EL1, Outer Shareable
AArch64	TLBI VAALE1	TLB Invalidate by VA, All ASID, Last level, EL1
AArch64	TLBI VAALE1IS	TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable
AArch64	TLBI VAALE1OS	TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable
AArch64	TLBI VAE1	TLB Invalidate by VA, EL1
AArch64	TLBI VAE1IS	TLB Invalidate by VA, EL1, Inner Shareable
AArch64	TLBI VAE1OS	TLB Invalidate by VA, EL1, Outer Shareable
AArch64	TLBI VAE2	TLB Invalidate by VA, EL2
AArch64	TLBI VAE2IS	TLB Invalidate by VA, EL2, Inner Shareable
AArch64	TLBI VAE2OS	TLB Invalidate by VA, EL2, Outer Shareable
AArch64	TLBI VAE3	TLB Invalidate by VA, EL3
AArch64	TLBI VAE3IS	TLB Invalidate by VA, EL3, Inner Shareable
AArch64	TLBI VAE3OS	TLB Invalidate by VA, EL3, Outer Shareable
AArch64	TLBI VALE1	TLB Invalidate by VA, Last level, EL1
AArch64	TLBI VALE1IS	TLB Invalidate by VA, Last level, EL1, Inner Shareable
AArch64	TLBI VALE1OS	TLB Invalidate by VA, Last level, EL1, Outer Shareable
AArch64	TLBI VALE2	TLB Invalidate by VA, Last level, EL2
AArch64	TLBI VALE2IS	TLB Invalidate by VA, Last level, EL2, Inner Shareable

Exec state	Name	Description
AArch64	TLBI VALE2OS	TLB Invalidate by VA, Last level, EL2, Outer Shareable
AArch64	TLBI VALE3	TLB Invalidate by VA, Last level, EL3
AArch64	TLBI VALE3IS	TLB Invalidate by VA, Last level, EL3, Inner Shareable
AArch64	TLBI VALE3OS	TLB Invalidate by VA, Last level, EL3, Outer Shareable
AArch64	TLBI VMALLE1	TLB Invalidate by VMID, All at stage 1, EL1
AArch64	TLBI VMALLE1IS	TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable
AArch64	TLBI VMALLE1OS	TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable
AArch64	TLBI VMALLS12E1	TLB Invalidate by VMID, All at Stage 1 and 2, EL1
AArch64	TLBI VMALLS12E1IS	TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable
AArch64	TLBI VMALLS12E1OS	TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable
AArch64	TLBI VMALLWS2E1	TLB Invalidate stage 2 dirty state by VMID, EL1&0
AArch64	TLBI VMALLWS2E1IS	TLB Invalidate stage 2 dirty state by VMID, EL1&0, Inner Shareable
AArch64	TLBI VMALLWS2E1OS	TLB Invalidate stage 2 write permission by VMID, EL1&0, Outer Shareable
AArch32	TLBIALL	TLB Invalidate All
AArch32	TLBIALLH	TLB Invalidate All, Hyp mode
AArch32	TLBIALLHIS	TLB Invalidate All, Hyp mode, Inner Shareable
AArch32	TLBIALLIS	TLB Invalidate All, Inner Shareable
AArch32	TLBIALLNSNH	TLB Invalidate All, Non-Secure Non-Hyp
AArch32	TLBIALLNSNHIS	TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable
AArch32	TLBIASID	TLB Invalidate by ASID match
AArch32	TLBIASIDIS	TLB Invalidate by ASID match, Inner Shareable
AArch32	TLBIIPAS2	TLB Invalidate by Intermediate Physical Address, Stage 2
AArch32	TLBIIPAS2IS	TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable
AArch32	TLBIIPAS2L	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level
AArch32	TLBIIPAS2LIS	TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable
AArch32	TLBIMVA	TLB Invalidate by VA
AArch32	TLBIMVAA	TLB Invalidate by VA, All ASID
AArch32	TLBIMVAAIS	TLB Invalidate by VA, All ASID, Inner Shareable
AArch32	TLBIMVAAL	TLB Invalidate by VA, All ASID, Last level
AArch32	TLBIMVAALIS	TLB Invalidate by VA, All ASID, Last level, Inner Shareable
AArch32	TLBIMVAH	TLB Invalidate by VA, Hyp mode
AArch32	TLBIMVAHIS	TLB Invalidate by VA, Hyp mode, Inner Shareable
AArch32	TLBIMVAIS	TLB Invalidate by VA, Inner Shareable
AArch32	TLBIMVAL	TLB Invalidate by VA, Last level
AArch32	TLBIMVALH	TLB Invalidate by VA, Last level, Hyp mode
AArch32	TLBIMVALHIS	TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable
AArch32	TLBIMVALIS	TLB Invalidate by VA, Last level, Inner Shareable
AArch64	TLBIP IPAS2E1	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBIP IPAS2E1IS	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBIP IPAS2E1OS	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBIP IPAS2LE1	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBIP IPAS2LE1IS	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBIP IPAS2LE1OS	TLB Invalidate Pair by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TLBIP RIPAS2E1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1
AArch64	TLBIP RIPAS2E1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable
AArch64	TLBIP RIPAS2E1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable
AArch64	TLBIP RIPAS2LE1	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64	TLBIP RIPAS2LE1IS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64	TLBIP RIPAS2LE1OS	TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64	TLBIP RVAAE1	TLB Range Invalidate by VA, All ASID, EL1
AArch64	TLBIP RVAAE1IS	TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable
AArch64	TLBIP RVAAE1OS	TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable
AArch64	TLBIP RVAALE1	TLB Range Invalidate by VA, All ASID, Last level, EL1
AArch64	TLBIP RVAALE1IS	TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable
AArch64	TLBIP RVAALE1OS	TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable
AArch64	TLBIP RVAE1	TLB Range Invalidate by VA, EL1
AArch64	TLBIP RVAE1IS	TLB Range Invalidate by VA, EL1, Inner Shareable
AArch64	TLBIP RVAE1OS	TLB Range Invalidate by VA, EL1, Outer Shareable
AArch64	TLBIP RVAE2	TLB Range Invalidate by VA, EL2
AArch64	TLBIP RVAE2IS	TLB Range Invalidate by VA, EL2, Inner Shareable
AArch64	TLBIP RVAE2OS	TLB Range Invalidate by VA, EL2, Outer Shareable
AArch64	TLBIP RVAE3	TLB Range Invalidate by VA, EL3
AArch64	TLBIP RVAE3IS	TLB Range Invalidate by VA, EL3, Inner Shareable
AArch64	TLBIP RVAE3OS	TLB Range Invalidate by VA, EL3, Outer Shareable

Exec state	Name	Description
AArch64	TLBIP RVALE1	TLB Range Invalidate by VA, Last level, EL1
AArch64	TLBIP RVALE1IS	TLB Range Invalidate by VA, Last level, EL1, Inner Shareable
AArch64	TLBIP RVALE1OS	TLB Range Invalidate by VA, Last level, EL1, Outer Shareable
AArch64	TLBIP RVALE2	TLB Range Invalidate by VA, Last level, EL2
AArch64	TLBIP RVALE2IS	TLB Range Invalidate by VA, Last level, EL2, Inner Shareable
AArch64	TLBIP RVALE2OS	TLB Range Invalidate by VA, Last level, EL2, Outer Shareable
AArch64	TLBIP RVALE3	TLB Range Invalidate by VA, Last level, EL3
AArch64	TLBIP RVALE3IS	TLB Range Invalidate by VA, Last level, EL3, Inner Shareable
AArch64	TLBIP RVALE3OS	TLB Range Invalidate by VA, Last level, EL3, Outer Shareable
AArch64	TLBIP VAAE1	TLB Invalidate Pair by VA, All ASID, EL1
AArch64	TLBIP VAAE1IS	TLB Invalidate Pair by VA, All ASID, EL1, Inner Shareable
AArch64	TLBIP VAAE1OS	TLB Invalidate Pair by VA, All ASID, EL1, Outer Shareable
AArch64	TLBIP VAALE1	TLB Invalidate Pair by VA, All ASID, Last level, EL1
AArch64	TLBIP VAALE1IS	TLB Invalidate Pair by VA, All ASID, Last Level, EL1, Inner Shareable
AArch64	TLBIP VAALE1OS	TLB Invalidate Pair by VA, All ASID, Last Level, EL1, Outer Shareable
AArch64	TLBIP VAE1	TLB Invalidate Pair by VA, EL1
AArch64	TLBIP VAE1IS	TLB Invalidate Pair by VA, EL1, Inner Shareable
AArch64	TLBIP VAE1OS	TLB Invalidate Pair by VA, EL1, Outer Shareable
AArch64	TLBIP VAE2	TLB Invalidate Pair by VA, EL2
AArch64	TLBIP VAE2IS	TLB Invalidate Pair by VA, EL2, Inner Shareable
AArch64	TLBIP VAE2OS	TLB Invalidate Pair by VA, EL2, Outer Shareable
AArch64	TLBIP VAE3	TLB Invalidate Pair by VA, EL3
AArch64	TLBIP VAE3IS	TLB Invalidate Pair by VA, EL3, Inner Shareable
AArch64	TLBIP VAE3OS	TLB Invalidate Pair by VA, EL3, Outer Shareable
AArch64	TLBIP VALE1	TLB Invalidate Pair by VA, Last level, EL1
AArch64	TLBIP VALE1IS	TLB Invalidate Pair by VA, Last level, EL1, Inner Shareable
AArch64	TLBIP VALE1OS	TLB Invalidate Pair by VA, Last level, EL1, Outer Shareable
AArch64	TLBIP VALE2	TLB Invalidate Pair by VA, Last level, EL2
AArch64	TLBIP VALE2IS	TLB Invalidate Pair by VA, Last level, EL2, Inner Shareable
AArch64	TLBIP VALE2OS	TLB Invalidate Pair by VA, Last level, EL2, Outer Shareable
AArch64	TLBIP VALE3	TLB Invalidate Pair by VA, Last level, EL3
AArch64	TLBIP VALE3IS	TLB Invalidate Pair by VA, Last level, EL3, Inner Shareable
AArch64	TLBIP VALE3OS	TLB Invalidate Pair by VA, Last level, EL3, Outer Shareable

In the Legacy functional group:

Exec state	Name	Description
AArch32	CP15DMB	Data Memory Barrier System instruction
AArch32	CP15DSB	Data Synchronization Barrier System instruction
AArch32	CP15ISB	Instruction Synchronization Barrier System instruction
AArch32	FCSEIDR	FCSE Process ID register
AArch32	JIDR	Jazelle ID Register
AArch32	JMCR	Jazelle Main Configuration Register
AArch32	JOSCR	Jazelle OS Control Register

In the Other functional group:

Exec state	Name	Description
AArch64	ACTLRMASK_EL1	Auxiliary Control Masking Register (EL1)
AArch64	ACTLRMASK_EL2	Auxiliary Control Masking Register (EL2)
AArch32	CPACR	Architectural Feature Access Control Register
AArch64	CPACRMASK_EL1	Architectural Feature Access Control Masking Register
AArch64	CPACR_EL1	Architectural Feature Access Control Register
AArch64	CPTRMASK_EL2	Architectural Feature Trap Masking Register
AArch32	SCTLR	System Control Register
AArch64	SCTLR2MASK_EL1	Extended System Control Masking Register (EL1)
AArch64	SCTLR2MASK_EL2	Extended System Control Masking Register (EL2)
AArch64	SCTLR2_EL1	System Control Register (EL1)
AArch64	SCTLR2_EL3	System Control Register (EL3)
AArch64	SCTLRMASK_EL1	System Control Masking Register (EL1)
AArch64	SCTLRMASK_EL2	System Control Masking Register (EL2)
AArch64	SCTLR_EL1	System Control Register (EL1)

Exec state	Name	Description
AArch64	SCTLR_EL3	System Control Register (EL3)
AArch64	SMCR_EL1	SME Control Register (EL1)
AArch64	SMCR_EL2	SME Control Register (EL2)
AArch64	SMCR_EL3	SME Control Register (EL3)
AArch64	SMPRIMAP_EL2	Streaming Mode Priority Mapping Register
AArch64	SMPRI_EL1	Streaming Mode Priority Register
AArch64	TCR2MASK_EL1	Extended Translation Control Masking Register (EL1)
AArch64	TCR2MASK_EL2	Extended Translation Control Masking Register (EL2)
AArch64	TCRMASK_EL1	Translation Control Masking Register (EL1)
AArch64	TCRMASK_EL2	Translation Control Masking Register (EL2)
AArch64	ZCR_EL1	SVE Control Register (EL1)
AArch64	ZCR_EL2	SVE Control Register (EL2)
AArch64	ZCR_EL3	SVE Control Register (EL3)

In the Debug functional group:

Exec state	Name	Description
AArch32	DBGAUTHSTATUS	Debug Authentication Status register
AArch64	DBGAUTHSTATUS_EL1	Debug Authentication Status Register
External	DBGAUTHSTATUS_EL1	Debug Authentication Status Register
AArch32	DBGBCR<n>	Debug Breakpoint Control Registers
AArch64	DBGBCR<n>_EL1	Debug Breakpoint Control Registers
External	DBGBCR<n>_EL1	Debug Breakpoint Control Registers
AArch32	DBGBVR<n>	Debug Breakpoint Value Registers
AArch64	DBGBVR<n>_EL1	Debug Breakpoint Value Registers
External	DBGBVR<n>_EL1	Debug Breakpoint Value Registers
AArch32	DBGBXVR<n>	Debug Breakpoint Extended Value Registers
AArch32	DBGCLAIMCLR	Debug CLAIM Tag Clear register
AArch64	DBGCLAIMCLR_EL1	Debug CLAIM Tag Clear Register
External	DBGCLAIMCLR_EL1	Debug CLAIM Tag Clear Register
AArch32	DBGCLAIMSET	Debug CLAIM Tag Set register
AArch64	DBGCLAIMSET_EL1	Debug CLAIM Tag Set Register
External	DBGCLAIMSET_EL1	Debug CLAIM Tag Set Register
AArch32	DBGDCCINT	DCC Interrupt Enable Register
AArch32	DBGDEVID	Debug Device ID register 0
AArch32	DBGDEVID1	Debug Device ID register 1
AArch32	DBGDEVID2	Debug Device ID register 2
AArch32	DBGDIDR	Debug ID Register
AArch32	DBGDRAR	Debug ROM Address Register
AArch32	DBGDSAR	Debug Self Address Register
AArch32	DBGDSCRext	Debug Status and Control Register, External View
AArch32	DBGDSCRint	Debug Status and Control Register, Internal View
AArch64	DBGDTRRX_EL0	Debug Data Transfer Register, Receive
External	DBGDTRRX_EL0	Debug Data Transfer Register, Receive
AArch32	DBGDTRRXext	Debug OS Lock Data Transfer Register, Receive, External View
AArch32	DBGDTRRXint	Debug Data Transfer Register, Receive
AArch64	DBGDTRTX_EL0	Debug Data Transfer Register, Transmit
External	DBGDTRTX_EL0	Debug Data Transfer Register, Transmit
AArch32	DBGDTRTXext	Debug OS Lock Data Transfer Register, Transmit
AArch32	DBGDTRTXint	Debug Data Transfer Register, Transmit
AArch64	DBGDTR_EL0	Debug Data Transfer Register, half-duplex
AArch32	DBGOSDLR	Debug OS Double Lock Register
AArch32	DBGOSECCR	Debug OS Lock Exception Catch Control Register
AArch32	DBGOSLAR	Debug OS Lock Access Register
AArch32	DBGOSLSR	Debug OS Lock Status Register
AArch32	DBGPRCR	Debug Power Control Register
AArch64	DBGPRCR_EL1	Debug Power Control Register
AArch32	DBGVCR	Debug Vector Catch Register
AArch64	DBGVCR32_EL2	Debug Vector Catch Register
AArch32	DBGWCR<n>	Debug Watchpoint Control Registers
AArch64	DBGWCR<n>_EL1	Debug Watchpoint Control Registers
External	DBGWCR<n>_EL1	Debug Watchpoint Control Registers
AArch32	DBGWFAR	Debug Watchpoint Fault Address Register

Exec state	Name	Description
AArch32	DBGWVR<n>	Debug Watchpoint Value Registers
AArch64	DBGWVR<n>_EL1	Debug Watchpoint Value Registers
External	DBGWVR<n>_EL1	Debug Watchpoint Value Registers
External	EDACR	External Debug Auxiliary Control Register
External	EDCIDR0	External Debug Component Identification Register 0
External	EDCIDR1	External Debug Component Identification Register 1
External	EDCIDR2	External Debug Component Identification Register 2
External	EDCIDR3	External Debug Component Identification Register 3
External	EDCIDSR	External Debug Context ID Sample Register
External	EDDEVAFF0	External Debug Device Affinity register 0
External	EDDEVAFF1	External Debug Device Affinity register 1
External	EDDEVARCH	External Debug Device Architecture Register
External	EDDEVID	External Debug Device ID register 0
External	EDDEVID1	External Debug Device ID Register 1
External	EDDEVID2	External Debug Device ID register 2
External	EDDEVTYPE	External Debug Device Type register
External	EDECCR	External Debug Exception Catch Control Register
External	EDECR	External Debug Execution Control Register
External	EDESR	External Debug Event Status Register
External	EDHSR	External Debug Halting Syndrome Register
External	EDITCTRL	External Debug Integration mode Control register
External	EDITR	External Debug Instruction Transfer Register
External	EDLAR	External Debug Lock Access Register
External	EDLSR	External Debug Lock Status Register
External	EDPCSR	External Debug Program Counter Sample Register
External	EDPIDR0	External Debug Peripheral Identification Register 0
External	EDPIDR1	External Debug Peripheral Identification Register 1
External	EDPIDR2	External Debug Peripheral Identification Register 2
External	EDPIDR3	External Debug Peripheral Identification Register 3
External	EDPIDR4	External Debug Peripheral Identification Register 4
External	EDPRCR	External Debug Power/Reset Control Register
External	EDPRSR	External Debug Processor Status Register
External	EDRCR	External Debug Reserve Control Register
External	EDSCR	External Debug Status and Control Register
External	EDSCR2	External Debug Status and Control Register 2
External	EDVIDSR	External Debug Virtual Context Sample Register
External	EDWAR	External Debug Watchpoint Address Register
AArch64	MDCCINT_EL1	Monitor DCC Interrupt Enable Register
AArch64	MDCCSR_EL0	Monitor DCC Status Register
AArch64	MDRAR_EL1	Monitor Debug ROM Address Register
AArch64	MDSCR_EL1	Monitor Debug System Control Register
AArch64	OSDLR_EL1	OS Double Lock Register
AArch64	OSDTRRX_EL1	OS Lock Data Transfer Register, Receive
AArch64	OSDTRTX_EL1	OS Lock Data Transfer Register, Transmit
AArch64	OSECCR_EL1	OS Lock Exception Catch Control Register
AArch64	OSLAR_EL1	OS Lock Access Register
External	OSLAR_EL1	OS Lock Access Register
AArch64	OSLSR_EL1	OS Lock Status Register
AArch32	TRFCR	Trace Filter Control Register
AArch64	TRFCR_EL1	Trace Filter Control Register (EL1)
AArch64	TRFCR_EL2	Trace Filter Control Register (EL2)

In the Special functional group:

Exec state	Name	Description
AArch32	DLR	Debug Link Register
AArch32	DSPSR	Debug Saved Program Status Register
AArch64	ELR_EL1	Exception Link Register (EL1)
AArch64	ELR_EL2	Exception Link Register (EL2)
AArch64	ELR_EL3	Exception Link Register (EL3)
AArch32	ELR_hyp	Exception Link Register (Hyp mode)
AArch32	SPSR	Saved Program Status Register
AArch64	SPSR_EL1	Saved Program Status Register (EL1)

Exec state	Name	Description
AArch64	SPSR_EL2	Saved Program Status Register (EL2)
AArch64	SPSR_EL3	Saved Program Status Register (EL3)
AArch32	SPSR_abt	Saved Program Status Register (Abort mode)
AArch64	SPSR_abt	Saved Program Status Register (Abort mode)
AArch32	SPSR_fiq	Saved Program Status Register (FIQ mode)
AArch64	SPSR_fiq	Saved Program Status Register (FIQ mode)
AArch32	SPSR_hyp	Saved Program Status Register (Hyp mode)
AArch32	SPSR_irq	Saved Program Status Register (IRQ mode)
AArch64	SPSR_irq	Saved Program Status Register (IRQ mode)
AArch32	SPSR_mon	Saved Program Status Register (Monitor mode)
AArch32	SPSR_svc	Saved Program Status Register (Supervisor mode)
AArch32	SPSR_und	Saved Program Status Register (Undefined mode)
AArch64	SPSR_und	Saved Program Status Register (Undefined mode)
AArch64	SP_EL0	Stack Pointer (EL0)
AArch64	SP_EL1	Stack Pointer (EL1)
AArch64	SP_EL2	Stack Pointer (EL2)
AArch64	SP_EL3	Stack Pointer (EL3)

In the Unknown functional group:

Exec state	Name	Description
AArch64	ACCDATA_EL1	Accelerator Data
AArch32	DSPSR2	Debug Saved Process State Register 2
AArch64	FGWTE3_EL3	Fine-Grained Write Traps EL3
AArch64	HAFGRTR_EL2	Hypervisor Activity Monitors Fine-Grained Read Trap Register
AArch64	HDFGRTR2_EL2	Hypervisor Debug Fine-Grained Read Trap Register 2
AArch64	HDFGRTR_EL2	Hypervisor Debug Fine-Grained Read Trap Register
AArch64	HDFGWTR2_EL2	Hypervisor Debug Fine-Grained Write Trap Register 2
AArch64	HDFGWTR_EL2	Hypervisor Debug Fine-Grained Write Trap Register
AArch64	HFGITR_EL2	Hypervisor Fine-Grained Instruction Trap Register
AArch64	HFGRTR2_EL2	Hypervisor Fine-Grained Read Trap Register 2
AArch64	HFGRTR_EL2	Hypervisor Fine-Grained Read Trap Register
AArch64	HFGWTR2_EL2	Hypervisor Fine-Grained Write Trap Register 2
AArch64	HFGWTR_EL2	Hypervisor Fine-Grained Write Trap Register
AArch64	MDSELR_EL1	Breakpoint and Watchpoint Selection Register
AArch64	MDSTEPOP_EL1	Monitor Debug Step Opcode Register
AArch64	PFAR_EL1	Physical Fault Address Register (EL1)
AArch64	PFAR_EL2	Physical Fault Address Register (EL2)
AArch64	PMICNTSVR_EL1	Performance Monitors Instruction Count Saved Value Register
AArch64	PMSSCR_EL1	Performance Monitors Snapshot Status and Capture Register
AArch64	SPMACCESSR_EL1	System Performance Monitors Access Register (EL1)
AArch64	SPMACCESSR_EL2	System Performance Monitors Access Register (EL2)
AArch64	SPMACCESSR_EL3	System Performance Monitors Access Register (EL3)
AArch64	SPMCFGR_EL1	System Performance Monitors Configuration Register
AArch64	SPMCGCR<n>_EL1	System PMU Counter Group Configuration Registers
AArch64	SPMCNTENCLR_EL0	System Performance Monitors Count Enable Clear Register
AArch64	SPMCNTENSET_EL0	System Performance Monitors Count Enable Set Register
AArch64	SPMCR_EL0	System Performance Monitor Control Register
AArch64	SPMDEVAFF_EL1	System Performance Monitors Device Affinity Register
AArch64	SPMDEVARCH_EL1	System Performance Monitors Device Architecture Register
AArch64	SPMEVCNTR<n>_EL0	System Performance Monitors Event Count Register
AArch64	SPMEVFILT2R<n>_EL0	System Performance Monitors Event Filter Control Register 2
AArch64	SPMEVFILTR<n>_EL0	System Performance Monitors Event Filter Control Register
AArch64	SPMEVTYPER<n>_EL0	System Performance Monitors Event Type Register
AArch64	SPMIIDR_EL1	System PMU Implementation Identification Register
AArch64	SPMINTENCLR_EL1	System Performance Monitors Interrupt Enable Clear Register
AArch64	SPMINTENSET_EL1	System Performance Monitors Interrupt Enable Set Register
AArch64	SPMOVSCLR_EL0	System Performance Monitors Overflow Flag Status Clear Register
AArch64	SPMOVSSET_EL0	System Performance Monitors Overflow Flag Status Set Register
AArch64	SPMROOTCR_EL3	System Performance Monitors Root and Realm Control Register
AArch64	SPMSCR_EL1	System Performance Monitors Secure Control Register
AArch64	SPMSELR_EL0	System Performance Monitors Select Register
AArch64	SPMZR_EL0	System Performance Monitors Zero with Mask

In the Float functional group:

Exec state	Name	Description
AArch64	FPCR	Floating-point Control Register
AArch32	FPEXC	Floating-Point Exception Control register
AArch64	FPEXC32_EL2	Floating-Point Exception Control Register
AArch64	FPMR	Floating-point Mode Register
AArch32	FPSCR	Floating-Point Status and Control Register
AArch32	FPSID	Floating-Point System ID register
AArch64	FPSR	Floating-point Status Register
AArch32	MVFR0	Media and VFP Feature Register 0
AArch64	MVFR0_EL1	AArch32 Media and VFP Feature Register 0
AArch32	MVFR1	Media and VFP Feature Register 1
AArch64	MVFR1_EL1	AArch32 Media and VFP Feature Register 1
AArch32	MVFR2	Media and VFP Feature Register 2
AArch64	MVFR2_EL1	AArch32 Media and VFP Feature Register 2

In the Reset Management functional group:

Exec state	Name	Description
AArch32	HRMR	Hyp Reset Management Register
AArch32	RMR	Reset Management Register
AArch64	RMR_EL1	Reset Management Register (EL1)
AArch64	RMR_EL2	Reset Management Register (EL2)
AArch64	RMR_EL3	Reset Management Register (EL3)
AArch32	RVBAR	Reset Vector Base Address Register
AArch64	RVBAR_EL1	Reset Vector Base Address Register (if EL2 and EL3 not implemented)
AArch64	RVBAR_EL2	Reset Vector Base Address Register (if EL3 not implemented)
AArch64	RVBAR_EL3	Reset Vector Base Address Register (if EL3 implemented)

In the Thread functional group:

Exec state	Name	Description
AArch32	HTPIDR	Hyp Software Thread ID Register
AArch64	SCXTNUM_EL0	EL0 Read/Write Software Context Number
AArch64	SCXTNUM_EL1	EL1 Read/Write Software Context Number
AArch64	SCXTNUM_EL2	EL2 Read/Write Software Context Number
AArch64	SCXTNUM_EL3	EL3 Read/Write Software Context Number
AArch64	TPIDR2_EL0	EL0 Read/Write Software Thread ID Register 2
AArch32	TPIDRPRW	PL1 Software Thread ID Register
AArch64	TPIDRRO_EL0	EL0 Read-Only Software Thread ID Register
AArch32	TPIDRURO	PL0 Read-Only Software Thread ID Register
AArch32	TPIDRURW	PL0 Read/Write Software Thread ID Register
AArch64	TPIDR_EL0	EL0 Read/Write Software Thread ID Register
AArch64	TPIDR_EL1	EL1 Software Thread ID Register
AArch64	TPIDR_EL2	EL2 Software Thread ID Register
AArch64	TPIDR_EL3	EL3 Software Thread ID Register

In the GIC control registers functional group:

Exec state	Name	Description
AArch32	ICC_AP0R<n>	Interrupt Controller Active Priorities Group 0 Registers
AArch64	ICC_AP0R<n>_EL1	Interrupt Controller Active Priorities Group 0 Registers
AArch32	ICC_AP1R<n>	Interrupt Controller Active Priorities Group 1 Registers
AArch64	ICC_AP1R<n>_EL1	Interrupt Controller Active Priorities Group 1 Registers
AArch32	ICC_ASGI1R	Interrupt Controller Alias Software Generated Interrupt Group 1 Register
AArch64	ICC_ASGI1R_EL1	Interrupt Controller Alias Software Generated Interrupt Group 1 Register
AArch32	ICC_BPR0	Interrupt Controller Binary Point Register 0
AArch64	ICC_BPR0_EL1	Interrupt Controller Binary Point Register 0
AArch32	ICC_BPR1	Interrupt Controller Binary Point Register 1
AArch64	ICC_BPR1_EL1	Interrupt Controller Binary Point Register 1

Exec state	Name	Description
AArch32	ICC_CTLR	Interrupt Controller Control Register
AArch64	ICC_CTLR_EL1	Interrupt Controller Control Register (EL1)
AArch64	ICC_CTLR_EL3	Interrupt Controller Control Register (EL3)
AArch32	ICC_DIR	Interrupt Controller Deactivate Interrupt Register
AArch64	ICC_DIR_EL1	Interrupt Controller Deactivate Interrupt Register
AArch32	ICC_EOIR0	Interrupt Controller End Of Interrupt Register 0
AArch64	ICC_EOIR0_EL1	Interrupt Controller End Of Interrupt Register 0
AArch32	ICC_EOIR1	Interrupt Controller End Of Interrupt Register 1
AArch64	ICC_EOIR1_EL1	Interrupt Controller End Of Interrupt Register 1
AArch32	ICC_HPPIR0	Interrupt Controller Highest Priority Pending Interrupt Register 0
AArch64	ICC_HPPIR0_EL1	Interrupt Controller Highest Priority Pending Interrupt Register 0
AArch32	ICC_HPPIR1	Interrupt Controller Highest Priority Pending Interrupt Register 1
AArch64	ICC_HPPIR1_EL1	Interrupt Controller Highest Priority Pending Interrupt Register 1
AArch32	ICC_HSRE	Interrupt Controller Hyp System Register Enable register
AArch32	ICC_IAR0	Interrupt Controller Interrupt Acknowledge Register 0
AArch64	ICC_IAR0_EL1	Interrupt Controller Interrupt Acknowledge Register 0
AArch32	ICC_IAR1	Interrupt Controller Interrupt Acknowledge Register 1
AArch64	ICC_IAR1_EL1	Interrupt Controller Interrupt Acknowledge Register 1
AArch32	ICC_IGRPEN0	Interrupt Controller Interrupt Group 0 Enable register
AArch64	ICC_IGRPEN0_EL1	Interrupt Controller Interrupt Group 0 Enable Register
AArch32	ICC_IGRPEN1	Interrupt Controller Interrupt Group 1 Enable register
AArch64	ICC_IGRPEN1_EL1	Interrupt Controller Interrupt Group 1 Enable Register
AArch64	ICC_IGRPEN1_EL3	Interrupt Controller Interrupt Group 1 Enable Register (EL3)
AArch32	ICC_MCTLR	Interrupt Controller Monitor Control Register
AArch32	ICC_MGRPEN1	Interrupt Controller Monitor Interrupt Group 1 Enable register
AArch32	ICC_MSRE	Interrupt Controller Monitor System Register Enable register
AArch64	ICC_NMIAR1_EL1	Interrupt Controller Non-maskable Interrupt Acknowledge Register 1
AArch32	ICC_PMR	Interrupt Controller Interrupt Priority Mask Register
AArch64	ICC_PMR_EL1	Interrupt Controller Interrupt Priority Mask Register
AArch32	ICC_RPR	Interrupt Controller Running Priority Register
AArch64	ICC_RPR_EL1	Interrupt Controller Running Priority Register
AArch32	ICC_SGI0R	Interrupt Controller Software Generated Interrupt Group 0 Register
AArch64	ICC_SGI0R_EL1	Interrupt Controller Software Generated Interrupt Group 0 Register
AArch32	ICC_SGI1R	Interrupt Controller Software Generated Interrupt Group 1 Register
AArch64	ICC_SGI1R_EL1	Interrupt Controller Software Generated Interrupt Group 1 Register
AArch32	ICC_SRE	Interrupt Controller System Register Enable register
AArch64	ICC_SRE_EL1	Interrupt Controller System Register Enable Register (EL1)
AArch64	ICC_SRE_EL2	Interrupt Controller System Register Enable Register (EL2)
AArch64	ICC_SRE_EL3	Interrupt Controller System Register Enable Register (EL3)

In the GIC functional group:

Exec state	Name	Description
AArch32	ICC_AP0R<n>	Interrupt Controller Active Priorities Group 0 Registers
AArch64	ICC_AP0R<n>_EL1	Interrupt Controller Active Priorities Group 0 Registers
AArch32	ICC_AP1R<n>	Interrupt Controller Active Priorities Group 1 Registers
AArch64	ICC_AP1R<n>_EL1	Interrupt Controller Active Priorities Group 1 Registers
AArch32	ICC_ASGI1R	Interrupt Controller Alias Software Generated Interrupt Group 1 Register
AArch64	ICC_ASGI1R_EL1	Interrupt Controller Alias Software Generated Interrupt Group 1 Register
AArch32	ICC_BPR0	Interrupt Controller Binary Point Register 0
AArch64	ICC_BPR0_EL1	Interrupt Controller Binary Point Register 0
AArch32	ICC_BPR1	Interrupt Controller Binary Point Register 1
AArch64	ICC_BPR1_EL1	Interrupt Controller Binary Point Register 1
AArch32	ICC_CTLR	Interrupt Controller Control Register
AArch64	ICC_CTLR_EL1	Interrupt Controller Control Register (EL1)
AArch64	ICC_CTLR_EL3	Interrupt Controller Control Register (EL3)
AArch32	ICC_DIR	Interrupt Controller Deactivate Interrupt Register
AArch64	ICC_DIR_EL1	Interrupt Controller Deactivate Interrupt Register
AArch32	ICC_EOIR0	Interrupt Controller End Of Interrupt Register 0
AArch64	ICC_EOIR0_EL1	Interrupt Controller End Of Interrupt Register 0
AArch32	ICC_EOIR1	Interrupt Controller End Of Interrupt Register 1
AArch64	ICC_EOIR1_EL1	Interrupt Controller End Of Interrupt Register 1
AArch32	ICC_HPPIR0	Interrupt Controller Highest Priority Pending Interrupt Register 0

Exec state	Name	Description
AArch64	ICC_HPPIR0_EL1	Interrupt Controller Highest Priority Pending Interrupt Register 0
AArch32	ICC_HPPIR1	Interrupt Controller Highest Priority Pending Interrupt Register 1
AArch64	ICC_HPPIR1_EL1	Interrupt Controller Highest Priority Pending Interrupt Register 1
AArch32	ICC_HSRE	Interrupt Controller Hyp System Register Enable register
AArch32	ICC_IAR0	Interrupt Controller Interrupt Acknowledge Register 0
AArch64	ICC_IAR0_EL1	Interrupt Controller Interrupt Acknowledge Register 0
AArch32	ICC_IAR1	Interrupt Controller Interrupt Acknowledge Register 1
AArch64	ICC_IAR1_EL1	Interrupt Controller Interrupt Acknowledge Register 1
AArch32	ICC_IGRPEN0	Interrupt Controller Interrupt Group 0 Enable register
AArch64	ICC_IGRPEN0_EL1	Interrupt Controller Interrupt Group 0 Enable Register
AArch32	ICC_IGRPEN1	Interrupt Controller Interrupt Group 1 Enable register
AArch64	ICC_IGRPEN1_EL1	Interrupt Controller Interrupt Group 1 Enable Register
AArch64	ICC_IGRPEN1_EL3	Interrupt Controller Interrupt Group 1 Enable Register (EL3)
AArch32	ICC_MCTLR	Interrupt Controller Monitor Control Register
AArch32	ICC_MGRPEN1	Interrupt Controller Monitor Interrupt Group 1 Enable register
AArch32	ICC_MSRE	Interrupt Controller Monitor System Register Enable register
AArch64	ICC_NMIAR1_EL1	Interrupt Controller Non-maskable Interrupt Acknowledge Register 1
AArch32	ICC_PMR	Interrupt Controller Interrupt Priority Mask Register
AArch64	ICC_PMR_EL1	Interrupt Controller Interrupt Priority Mask Register
AArch32	ICC_RPR	Interrupt Controller Running Priority Register
AArch64	ICC_RPR_EL1	Interrupt Controller Running Priority Register
AArch32	ICC_SGI0R	Interrupt Controller Software Generated Interrupt Group 0 Register
AArch64	ICC_SGI0R_EL1	Interrupt Controller Software Generated Interrupt Group 0 Register
AArch32	ICC_SGI1R	Interrupt Controller Software Generated Interrupt Group 1 Register
AArch64	ICC_SGI1R_EL1	Interrupt Controller Software Generated Interrupt Group 1 Register
AArch32	ICC_SRE	Interrupt Controller System Register Enable register
AArch64	ICC_SRE_EL1	Interrupt Controller System Register Enable Register (EL1)
AArch64	ICC_SRE_EL2	Interrupt Controller System Register Enable Register (EL2)
AArch64	ICC_SRE_EL3	Interrupt Controller System Register Enable Register (EL3)
AArch32	ICH_AP0R<n>	Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch64	ICH_AP0R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch32	ICH_AP1R<n>	Interrupt Controller Hyp Active Priorities Group 1 Registers
AArch64	ICH_AP1R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 1 Registers
AArch32	ICH_EISR	Interrupt Controller End of Interrupt Status Register
AArch64	ICH_EISR_EL2	Interrupt Controller End of Interrupt Status Register
AArch32	ICH_ELRSR	Interrupt Controller Empty List Register Status Register
AArch64	ICH_ELRSR_EL2	Interrupt Controller Empty List Register Status Register
AArch32	ICH_HCR	Interrupt Controller Hyp Control Register
AArch64	ICH_HCR_EL2	Interrupt Controller Hyp Control Register
AArch32	ICH_LR<n>	Interrupt Controller List Registers
AArch64	ICH_LR<n>_EL2	Interrupt Controller List Registers
AArch32	ICH_LRC<n>	Interrupt Controller List Registers
AArch32	ICH_MISR	Interrupt Controller Maintenance Interrupt State Register
AArch64	ICH_MISR_EL2	Interrupt Controller Maintenance Interrupt State Register
AArch32	ICH_VMCR	Interrupt Controller Virtual Machine Control Register
AArch64	ICH_VMCR_EL2	Interrupt Controller Virtual Machine Control Register
AArch32	ICH_VTR	Interrupt Controller VGIC Type Register
AArch64	ICH_VTR_EL2	Interrupt Controller VGIC Type Register
AArch32	ICV_AP0R<n>	Interrupt Controller Virtual Active Priorities Group 0 Registers
AArch64	ICV_AP0R<n>_EL1	Interrupt Controller Virtual Active Priorities Group 0 Registers
AArch32	ICV_AP1R<n>	Interrupt Controller Virtual Active Priorities Group 1 Registers
AArch64	ICV_AP1R<n>_EL1	Interrupt Controller Virtual Active Priorities Group 1 Registers
AArch32	ICV_BPR0	Interrupt Controller Virtual Binary Point Register 0
AArch64	ICV_BPR0_EL1	Interrupt Controller Virtual Binary Point Register 0
AArch32	ICV_BPR1	Interrupt Controller Virtual Binary Point Register 1
AArch64	ICV_BPR1_EL1	Interrupt Controller Virtual Binary Point Register 1
AArch32	ICV_CTLR	Interrupt Controller Virtual Control Register
AArch64	ICV_CTLR_EL1	Interrupt Controller Virtual Control Register
AArch32	ICV_DIR	Interrupt Controller Deactivate Virtual Interrupt Register
AArch64	ICV_DIR_EL1	Interrupt Controller Deactivate Virtual Interrupt Register
AArch32	ICV_EOIR0	Interrupt Controller Virtual End Of Interrupt Register 0
AArch64	ICV_EOIR0_EL1	Interrupt Controller Virtual End Of Interrupt Register 0
AArch32	ICV_EOIR1	Interrupt Controller Virtual End Of Interrupt Register 1
AArch64	ICV_EOIR1_EL1	Interrupt Controller Virtual End Of Interrupt Register 1

Exec state	Name	Description
AArch32	ICV_HPPIR0	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0
AArch64	ICV_HPPIR0_EL1	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0
AArch32	ICV_HPPIR1	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1
AArch64	ICV_HPPIR1_EL1	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1
AArch32	ICV_IAR0	Interrupt Controller Virtual Interrupt Acknowledge Register 0
AArch64	ICV_IAR0_EL1	Interrupt Controller Virtual Interrupt Acknowledge Register 0
AArch32	ICV_IAR1	Interrupt Controller Virtual Interrupt Acknowledge Register 1
AArch64	ICV_IAR1_EL1	Interrupt Controller Virtual Interrupt Acknowledge Register 1
AArch32	ICV_IGRPEN0	Interrupt Controller Virtual Interrupt Group 0 Enable register
AArch64	ICV_IGRPEN0_EL1	Interrupt Controller Virtual Interrupt Group 0 Enable Register
AArch32	ICV_IGRPEN1	Interrupt Controller Virtual Interrupt Group 1 Enable register
AArch64	ICV_IGRPEN1_EL1	Interrupt Controller Virtual Interrupt Group 1 Enable Register
AArch64	ICV_NMIAR1_EL1	Interrupt Controller Virtual Non-maskable Interrupt Acknowledge Register 1
AArch32	ICV_PMR	Interrupt Controller Virtual Interrupt Priority Mask Register
AArch64	ICV_PMR_EL1	Interrupt Controller Virtual Interrupt Priority Mask Register
AArch32	ICV_RPR	Interrupt Controller Virtual Running Priority Register
AArch64	ICV_RPR_EL1	Interrupt Controller Virtual Running Priority Register

In the Secure functional group:

Exec state	Name	Description
AArch64	ACTLR_EL3	Auxiliary Control Register (EL3)
AArch64	AFSR0_EL3	Auxiliary Fault Status Register 0 (EL3)
AArch64	AFSR1_EL3	Auxiliary Fault Status Register 1 (EL3)
AArch64	AMAIR_EL3	Auxiliary Memory Attribute Indirection Register (EL3)
AArch64	CPTR_EL3	Architectural Feature Trap Register (EL3)
AArch64	ICC_CTLR_EL3	Interrupt Controller Control Register (EL3)
AArch32	ICC_MCTLR	Interrupt Controller Monitor Control Register
AArch32	ICC_MSRE	Interrupt Controller Monitor System Register Enable register
AArch64	ICC_SRE_EL3	Interrupt Controller System Register Enable Register (EL3)
AArch64	MDCR_EL3	Monitor Debug Configuration Register (EL3)
AArch32	MVBAR	Monitor Vector Base Address Register
AArch32	NSACR	Non-Secure Access Control Register
AArch32	SCR	Secure Configuration Register
AArch64	SCR_EL3	Secure Configuration Register
AArch32	SDCR	Secure Debug Control Register
AArch32	SDER	Secure Debug Enable Register
AArch64	SDER32_EL3	AArch32 Secure Debug Enable Register
AArch64	VBAR_EL3	Vector Base Address Register (EL3)

In the GIC Host Interface Control Registers functional group:

Exec state	Name	Description
AArch32	ICH_AP0R<n>	Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch64	ICH_AP0R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch32	ICH_AP1R<n>	Interrupt Controller Hyp Active Priorities Group 1 Registers
AArch64	ICH_AP1R<n>_EL2	Interrupt Controller Hyp Active Priorities Group 1 Registers
AArch32	ICH_EISR	Interrupt Controller End of Interrupt Status Register
AArch64	ICH_EISR_EL2	Interrupt Controller End of Interrupt Status Register
AArch32	ICH_ELRSR	Interrupt Controller Empty List Register Status Register
AArch64	ICH_ELRSR_EL2	Interrupt Controller Empty List Register Status Register
AArch32	ICH_HCR	Interrupt Controller Hyp Control Register
AArch64	ICH_HCR_EL2	Interrupt Controller Hyp Control Register
AArch32	ICH_LR<n>	Interrupt Controller List Registers
AArch64	ICH_LR<n>_EL2	Interrupt Controller List Registers
AArch32	ICH_LRC<n>	Interrupt Controller List Registers
AArch32	ICH_MISR	Interrupt Controller Maintenance Interrupt State Register
AArch64	ICH_MISR_EL2	Interrupt Controller Maintenance Interrupt State Register
AArch32	ICH_VMCR	Interrupt Controller Virtual Machine Control Register
AArch64	ICH_VMCR_EL2	Interrupt Controller Virtual Machine Control Register
AArch32	ICH_VTR	Interrupt Controller VGIC Type Register
AArch64	ICH_VTR_EL2	Interrupt Controller VGIC Type Register

In the GICV Control functional group:

Exec state	Name	Description
AArch32	ICV_AP0R<n>	Interrupt Controller Virtual Active Priorities Group 0 Registers
AArch64	ICV_AP0R<n>_EL1	Interrupt Controller Virtual Active Priorities Group 0 Registers
AArch32	ICV_AP1R<n>	Interrupt Controller Virtual Active Priorities Group 1 Registers
AArch64	ICV_AP1R<n>_EL1	Interrupt Controller Virtual Active Priorities Group 1 Registers
AArch32	ICV_BPR0	Interrupt Controller Virtual Binary Point Register 0
AArch64	ICV_BPR0_EL1	Interrupt Controller Virtual Binary Point Register 0
AArch32	ICV_BPR1	Interrupt Controller Virtual Binary Point Register 1
AArch64	ICV_BPR1_EL1	Interrupt Controller Virtual Binary Point Register 1
AArch32	ICV_CTLR	Interrupt Controller Virtual Control Register
AArch64	ICV_CTLR_EL1	Interrupt Controller Virtual Control Register
AArch32	ICV_DIR	Interrupt Controller Deactivate Virtual Interrupt Register
AArch64	ICV_DIR_EL1	Interrupt Controller Deactivate Virtual Interrupt Register
AArch32	ICV_EOIR0	Interrupt Controller Virtual End Of Interrupt Register 0
AArch64	ICV_EOIR0_EL1	Interrupt Controller Virtual End Of Interrupt Register 0
AArch32	ICV_EOIR1	Interrupt Controller Virtual End Of Interrupt Register 1
AArch64	ICV_EOIR1_EL1	Interrupt Controller Virtual End Of Interrupt Register 1
AArch32	ICV_HPPIR0	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0
AArch64	ICV_HPPIR0_EL1	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0
AArch32	ICV_HPPIR1	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1
AArch64	ICV_HPPIR1_EL1	Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1
AArch32	ICV_IAR0	Interrupt Controller Virtual Interrupt Acknowledge Register 0
AArch64	ICV_IAR0_EL1	Interrupt Controller Virtual Interrupt Acknowledge Register 0
AArch32	ICV_IAR1	Interrupt Controller Virtual Interrupt Acknowledge Register 1
AArch64	ICV_IAR1_EL1	Interrupt Controller Virtual Interrupt Acknowledge Register 1
AArch32	ICV_IGRPEN0	Interrupt Controller Virtual Interrupt Group 0 Enable register
AArch64	ICV_IGRPEN0_EL1	Interrupt Controller Virtual Interrupt Group 0 Enable Register
AArch32	ICV_IGRPEN1	Interrupt Controller Virtual Interrupt Group 1 Enable register
AArch64	ICV_IGRPEN1_EL1	Interrupt Controller Virtual Interrupt Group 1 Enable Register
AArch64	ICV_NMIAR1_EL1	Interrupt Controller Virtual Non-maskable Interrupt Acknowledge Register 1
AArch32	ICV_PMR	Interrupt Controller Virtual Interrupt Priority Mask Register
AArch64	ICV_PMR_EL1	Interrupt Controller Virtual Interrupt Priority Mask Register
AArch32	ICV_RPR	Interrupt Controller Virtual Running Priority Register
AArch64	ICV_RPR_EL1	Interrupt Controller Virtual Running Priority Register

In the Generic System Control functional group:

Exec state	Name	Description
AArch64	GCR_EL1	Tag Control Register.
AArch64	MECIDR_EL2	MEC Identification Register
AArch64	MECID_A0_EL2	Alternate MECID for EL2 and EL2&0 translation regimes
AArch64	MECID_A1_EL2	Alternate MECID for EL2&0 translation regimes.
AArch64	MECID_P0_EL2	Primary MECID for EL2 and EL2&0 translation regimes
AArch64	MECID_P1_EL2	Primary MECID for EL2&0 translation regimes
AArch64	MECID_RL_A_EL3	Realm PA space Alternate MECID for EL3 stage 1 translation regime
AArch32	PAR	Physical Address Register
AArch64	PAR_EL1	Physical Address Register
AArch64	RGSR_EL1	Random Allocation Tag Seed Register.
AArch64	RNDR	Random Number
AArch64	RNDRRS	Random Number Full Entropy
AArch64	SDER32_EL2	AArch32 Secure Debug Enable Register
AArch64	TFSRE0_EL1	Tag Fault Status Register (EL0).
AArch64	TFSR_EL1	Tag Fault Status Register (EL1)
AArch64	TFSR_EL2	Tag Fault Status Register (EL2)
AArch64	TFSR_EL3	Tag Fault Status Register (EL3)
AArch64	VMECID_A_EL2	Alternate MECID for EL1&0 stage 2 translation regime
AArch64	VMECID_P_EL2	Primary MECID for EL1&0 stage 2 translation regime
AArch64	VNCR_EL2	Virtual Nested Control Register
AArch64	VSTCR_EL2	Virtualization Secure Translation Control Register
AArch64	VSTTBR_EL2	Virtualization Secure Translation Table Base Register

In the PMU functional group:

Exec state	Name	Description
External	PMAUTHSTATUS	Performance Monitors Authentication Status register
AArch32	PMCCFILTR	Performance Monitors Cycle Count Filter Register
AArch64	PMCCFILTR_EL0	Performance Monitors Cycle Count Filter Register
External	PMCCFILTR_EL0	Performance Monitors Cycle Counter Filter Register
External	PMCCIDSR	CONTEXTIDR_ELx Sample Register
AArch32	PMCCNTR	Performance Monitors Cycle Count Register
AArch64	PMCCNTR_EL0	Performance Monitors Cycle Count Register
External	PMCCNTR_EL0	Performance Monitors Cycle Counter
AArch64	PMCCNTSVR_EL1	Performance Monitors Cycle Count Saved Value Register
External	PMCCNTSVR_EL1	Performance Monitors Cycle Count Saved Value Register
External	PMCCR	PMU Configuration Control Register
AArch32	PMCEID0	Performance Monitors Common Event Identification register 0
External	PMCEID0	Performance Monitors Common Event Identification register 0
AArch64	PMCEID0_EL0	Performance Monitors Common Event Identification Register 0
AArch32	PMCEID1	Performance Monitors Common Event Identification register 1
External	PMCEID1	Performance Monitors Common Event Identification register 1
AArch64	PMCEID1_EL0	Performance Monitors Common Event Identification Register 1
AArch32	PMCEID2	Performance Monitors Common Event Identification register 2
External	PMCEID2	Performance Monitors Common Event Identification register 2
AArch32	PMCEID3	Performance Monitors Common Event Identification register 3
External	PMCEID3	Performance Monitors Common Event Identification register 3
External	PMCFGR	Performance Monitors Configuration Register
External	PMCGCR0	Counter Group Configuration Register 0
External	PMCID1SR	CONTEXTIDR_EL1 Sample Register
External	PMCID2SR	CONTEXTIDR_EL2 Sample Register
External	PMCIDR0	Performance Monitors Component Identification Register 0
External	PMCIDR1	Performance Monitors Component Identification Register 1
External	PMCIDR2	Performance Monitors Component Identification Register 2
External	PMCIDR3	Performance Monitors Component Identification Register 3
External	PMCNTEN	Performance Monitors Count Enable register
AArch32	PMCNTENCLR	Performance Monitors Count Enable Clear register
AArch64	PMCNTENCLR_EL0	Performance Monitors Count Enable Clear Register
External	PMCNTENCLR_EL0	Performance Monitors Count Enable Clear Register
AArch32	PMCNTENSET	Performance Monitors Count Enable Set register
AArch64	PMCNTENSET_EL0	Performance Monitors Count Enable Set Register
External	PMCNTENSET_EL0	Performance Monitors Count Enable Set Register
AArch32	PMCR	Performance Monitors Control Register
AArch64	PMCR_EL0	Performance Monitors Control Register
External	PMCR_EL0	Performance Monitors Control Register
External	PMDEVAFF	Performance Monitors Device Affinity register
External	PMDEVAFF0	Performance Monitors Device Affinity register 0
External	PMDEVAFF1	Performance Monitors Device Affinity register 1
External	PMDEVARCH	Performance Monitors Device Architecture register
External	PMDEVID	Performance Monitors Device ID register
External	PMDEVTYPE	Performance Monitors Device Type register
AArch64	PMECR_EL1	Performance Monitors Extended Control Register (EL1)
AArch32	PMEVCNTR<n>	Performance Monitors Event Count Registers
AArch64	PMEVCNTR<n>_EL0	Performance Monitors Event Count Registers
External	PMEVCNTR<n>_EL0	Performance Monitors Event Count Registers
AArch64	PMEVCNTSVR<n>_EL1	Performance Monitors Event Count Saved Value Registers
External	PMEVCNTSVR<n>_EL1	Performance Monitors Event Count Saved Value Registers
External	PMEVFILT2R<n>	Performance Monitors Event Filter Registers
AArch32	PMEVTYPER<n>	Performance Monitors Event Type Registers
AArch64	PMEVTYPER<n>_EL0	Performance Monitors Event Type Registers
External	PMEVTYPER<n>_EL0	Performance Monitors Event Type Registers
AArch64	PMIAR_EL1	Performance Monitors Instruction Address Register
AArch64	PMICFILTR_EL0	Performance Monitors Instruction Counter Filter Register
External	PMICFILTR_EL0	Performance Monitors Instruction Counter Filter Register
AArch64	PMICNTR_EL0	Performance Monitors Instruction Counter Register
External	PMICNTR_EL0	Performance Monitors Instruction Counter Register
External	PMICNTSVR_EL1	Performance Monitors Instruction Count Saved Value Register
External	PMIIDR	Performance Monitors Implementation Identification Register

Exec state	Name	Description
External	PMINTEN	Performance Monitors Interrupt Enable register
AArch32	PMINTENCLR	Performance Monitors Interrupt Enable Clear register
AArch64	PMINTENCLR_EL1	Performance Monitors Interrupt Enable Clear Register
External	PMINTENCLR_EL1	Performance Monitors Interrupt Enable Clear Register
AArch32	PMINTENSET	Performance Monitors Interrupt Enable Set register
AArch64	PMINTENSET_EL1	Performance Monitors Interrupt Enable Set Register
External	PMINTENSET_EL1	Performance Monitors Interrupt Enable Set Register
External	PMITCTRL	Performance Monitors Integration mode Control register
External	PMLAR	Performance Monitors Lock Access Register
External	PMLSR	Performance Monitors Lock Status Register
AArch32	PMMIR	Performance Monitors Machine Identification Register
External	PMMIR	Performance Monitors Machine Identification Register
AArch64	PMMIR_EL1	Performance Monitors Machine Identification Register
External	PMOVS	Performance Monitors Overflow Flag Status register
AArch64	PMOVSCLR_EL0	Performance Monitors Overflow Flag Status Clear Register
External	PMOVSCLR_EL0	Performance Monitors Overflow Flag Status Clear register
AArch32	PMOVSR	Performance Monitors Overflow Flag Status Register
AArch32	PMOVSSET	Performance Monitors Overflow Flag Status Set register
AArch64	PMOVSSET_EL0	Performance Monitors Overflow Flag Status Set Register
External	PMOVSSET_EL0	Performance Monitors Overflow Flag Status Set Register
External	PMPCSCTL	PC Sample-based Profiling Control Register
External	PMPCSR	Program Counter Sample Register
External	PMPIDR0	Performance Monitors Peripheral Identification Register 0
External	PMPIDR1	Performance Monitors Peripheral Identification Register 1
External	PMPIDR2	Performance Monitors Peripheral Identification Register 2
External	PMPIDR3	Performance Monitors Peripheral Identification Register 3
External	PMPIDR4	Performance Monitors Peripheral Identification Register 4
AArch32	PMSELR	Performance Monitors Event Counter Selection Register
AArch64	PMSELR_EL0	Performance Monitors Event Counter Selection Register
External	PMSSCR_EL1	Performance Monitors Snapshot Status and Capture Register
AArch32	PMSWINC	Performance Monitors Software Increment register
AArch64	PMSWINC_EL0	Performance Monitors Software Increment Register
External	PMSWINC_EL0	Performance Monitors Software Increment Register
AArch64	PMUACR_EL1	Performance Monitors User Access Control Register
AArch32	PMUSERENR	Performance Monitors User Enable Register
AArch64	PMUSERENR_EL0	Performance Monitors User Enable Register
External	PMVCIDSR	CONTEXTIDR_EL1 and VMID Sample Register
External	PMVIDSR	VMID Sample Register
AArch32	PMXEVCNTR	Performance Monitors Selected Event Count Register
AArch64	PMXEVCNTR_EL0	Performance Monitors Selected Event Count Register
AArch32	PMXEVTYPER	Performance Monitors Selected Event Type Register
AArch64	PMXEVTYPER_EL0	Performance Monitors Selected Event Type Register
AArch64	PMZR_EL0	Performance Monitors Zero with Mask
External	PMZR_EL0	Performance Monitors Zero with Mask

In the Root functional group:

Exec state	Name	Description
AArch64	APAS	Associate PA space
AArch64	GPCBW_EL3	Granule Protection Check Bypass Window Register (EL3)
AArch64	GPCCR_EL3	Granule Protection Check Control Register (EL3)
AArch64	GPTBR_EL3	Granule Protection Table Base Register

In the Pointer authentication functional group:

Exec state	Name	Description
AArch64	APDAKeyHi_EL1	Pointer Authentication Key A for Data (bits[127:64])
AArch64	APDAKeyLo_EL1	Pointer Authentication Key A for Data (bits[63:0])
AArch64	APDBKeyHi_EL1	Pointer Authentication Key B for Data (bits[127:64])
AArch64	APDBKeyLo_EL1	Pointer Authentication Key B for Data (bits[63:0])
AArch64	APGAKeyHi_EL1	Pointer Authentication Key A for Code (bits[127:64])
AArch64	APGAKeyLo_EL1	Pointer Authentication Key A for Code (bits[63:0])

Exec state	Name	Description
AArch64	APIAKeyHi_EL1	Pointer Authentication Key A for Instruction (bits[127:64])
AArch64	APIAKeyLo_EL1	Pointer Authentication Key A for Instruction (bits[63:0])
AArch64	APIBKeyHi_EL1	Pointer Authentication Key B for Instruction (bits[127:64])
AArch64	APIBKeyLo_EL1	Pointer Authentication Key B for Instruction (bits[63:0])

In the BRBE Instructions functional group:

Exec state	Name	Description
AArch64	BRB IALL	Invalidate the Branch Record Buffer
AArch64	BRB INJ	Branch Record Injection into the Branch Record Buffer

In the Debug Secondary functional group:

Exec state	Name	Description
AArch64	DLR_EL0	Debug Link Register
AArch64	DSPSR_EL0	Debug Saved Program Status Register

In the GCSE Instructions functional group:

Exec state	Name	Description
AArch64	GCSPOPCX	Guarded Control Stack Pop and Compare exception return record
AArch64	GCSPOPM	Guarded Control Stack Pop
AArch64	GCSPOPX	Guarded Control Stack Pop exception return record
AArch64	GCSPUSHM	Guarded Control Stack Push
AArch64	GCSPUSHX	Guarded Control Stack Push exception return record
AArch64	GCSSS1	Guarded Control Stack Switch Stack 1
AArch64	GCSSS2	Guarded Control Stack Switch Stack 2

In the RAS functional group:

Exec state	Name	Description
AArch32	DISR	Deferred Interrupt Status Register
AArch64	DISR_EL1	Deferred Interrupt Status Register
External	ERR<n>ADDR	Error Record <n> Address Register
External	ERR<n>CTLR	Error Record <n> Control Register
External	ERR<n>FR	Error Record <n> Feature Register
External	ERR<n>MISC0	Error Record <n> Miscellaneous Register 0
External	ERR<n>MISC1	Error Record <n> Miscellaneous Register 1
External	ERR<n>MISC2	Error Record <n> Miscellaneous Register 2
External	ERR<n>MISC3	Error Record <n> Miscellaneous Register 3
External	ERR<n>PFGCDN	Error Record <n> Pseudo-fault Generation Countdown Register
External	ERR<n>PFGCTL	Error Record <n> Pseudo-fault Generation Control Register
External	ERR<n>PFGF	Error Record <n> Pseudo-fault Generation Feature Register
External	ERR<n>STATUS	Error Record <n> Primary Status Register
External	ERRACR	Access Configuration Register
External	ERRCIDR0	Component Identification Register 0
External	ERRCIDR1	Component Identification Register 1
External	ERRCIDR2	Component Identification Register 2
External	ERRCIDR3	Component Identification Register 3
External	ERRCRICR0	Critical Error Interrupt Configuration Register 0
External	ERRCRICR1	Critical Error Interrupt Configuration Register 1
External	ERRCRICR2	Critical Error Interrupt Configuration Register 2
External	ERRDEVAFF	Device Affinity Register
External	ERRDEVARCH	Device Architecture Register
External	ERRDEVID	Device Configuration Register
External	ERRERICR0	Error Recovery Interrupt Configuration Register 0
External	ERRERICR1	Error Recovery Interrupt Configuration Register 1
External	ERRERICR2	Error Recovery Interrupt Configuration Register 2
External	ERRFHICR0	Fault Handling Interrupt Configuration Register 0
External	ERRFHICR1	Fault Handling Interrupt Configuration Register 1

Exec state	Name	Description
External	ERRFHICR2	Fault Handling Interrupt Configuration Register 2
External	ERRGSR<m>	Error Group <n> Status Register
AArch32	ERRIDR	Error Record ID Register
AArch64	ERRIDR_EL1	Error Record ID Register
External	ERRIIDR	Implementation Identification Register
External	ERRIMPDEF<n>	IMPLEMENTATION DEFINED Register <n>
External	ERRIRQCR<n>	Generic Error Interrupt Configuration Register <n>
External	ERRIRQSR	Error Interrupt Status Register
External	ERRPIDR0	Peripheral Identification Register 0
External	ERRPIDR1	Peripheral Identification Register 1
External	ERRPIDR2	Peripheral Identification Register 2
External	ERRPIDR3	Peripheral Identification Register 3
External	ERRPIDR4	Peripheral Identification Register 4
AArch32	ERRSELR	Error Record Select Register
AArch64	ERRSELR_EL1	Error Record Select Register
AArch32	ERXADDR	Selected Error Record Address Register
AArch32	ERXADDR2	Selected Error Record Address Register 2
AArch64	ERXADDR_EL1	Selected Error Record Address Register
AArch32	ERXCTLR	Selected Error Record Control Register
AArch32	ERXCTLR2	Selected Error Record Control Register 2
AArch64	ERXCTLR_EL1	Selected Error Record Control Register
AArch32	ERXFR	Selected Error Record Feature Register
AArch32	ERXFR2	Selected Error Record Feature Register 2
AArch64	ERXFR_EL1	Selected Error Record Feature Register
AArch64	ERXGSR_EL1	Selected Error Record Group Status Register
AArch32	ERXMISC0	Selected Error Record Miscellaneous Register 0
AArch64	ERXMISC0_EL1	Selected Error Record Miscellaneous Register 0
AArch32	ERXMISC1	Selected Error Record Miscellaneous Register 1
AArch64	ERXMISC1_EL1	Selected Error Record Miscellaneous Register 1
AArch32	ERXMISC2	Selected Error Record Miscellaneous Register 2
AArch64	ERXMISC2_EL1	Selected Error Record Miscellaneous Register 2
AArch32	ERXMISC3	Selected Error Record Miscellaneous Register 3
AArch64	ERXMISC3_EL1	Selected Error Record Miscellaneous Register 3
AArch32	ERXMISC4	Selected Error Record Miscellaneous Register 4
AArch32	ERXMISC5	Selected Error Record Miscellaneous Register 5
AArch32	ERXMISC6	Selected Error Record Miscellaneous Register 6
AArch32	ERXMISC7	Selected Error Record Miscellaneous Register 7
AArch64	ERXPFGCDN_EL1	Selected Pseudo-fault Generation Countdown Register
AArch64	ERXPFGCTL_EL1	Selected Pseudo-fault Generation Control Register
AArch64	ERXPFGF_EL1	Selected Pseudo-fault Generation Feature Register
AArch32	ERXSTATUS	Selected Error Record Primary Status Register
AArch64	ERXSTATUS_EL1	Selected Error Record Primary Status Register
AArch64	MFAR_EL3	Physical Fault Address Register (EL3)
AArch32	VDFSR	Virtual SError Exception Syndrome Register
AArch32	VDISR	Virtual Deferred Interrupt Status Register
AArch64	VDISR_EL2	Virtual Deferred Interrupt Status Register (EL2)
AArch64	VDISR_EL3	Virtual Deferred Interrupt Status Register (EL3)
AArch64	VSESR_EL2	Virtual SError Exception Syndrome Register (EL2)
AArch64	VSESR_EL3	Virtual SError Exception Syndrome Register (EL3)

In the Trace Unit Instructions functional group:

Exec state	Name	Description
AArch64	TRCIT	Trace Instrumentation

In the Trace functional group:

Exec state	Name	Description
AArch64	TRCACATR<n>	Trace Address Comparator Access Type Register <n>
External	TRCACATR<n>	Trace Address Comparator Access Type Register <n>
AArch64	TRCACVR<n>	Trace Address Comparator Value Register <n>
External	TRCACVR<n>	Trace Address Comparator Value Register <n>

Exec state	Name	Description
AArch64	TRCAUXCTLR	Trace Auxiliary Control Register
External	TRCAUXCTLR	Trace Auxiliary Control Register
AArch64	TRCBBCTLR	Trace Branch Broadcast Control Register
External	TRCBBCTLR	Trace Branch Broadcast Control Register
AArch64	TRCCCCTLR	Trace Cycle Count Control Register
External	TRCCCCTLR	Trace Cycle Count Control Register
AArch64	TRCCIDCCTLR0	Trace Context Identifier Comparator Control Register 0
External	TRCCIDCCTLR0	Trace Context Identifier Comparator Control Register 0
AArch64	TRCCIDCCTLR1	Trace Context Identifier Comparator Control Register 1
External	TRCCIDCCTLR1	Trace Context Identifier Comparator Control Register 1
AArch64	TRCCIDCVR<n>	Trace Context Identifier Comparator Value Registers <n>
External	TRCCIDCVR<n>	Trace Context Identifier Comparator Value Registers <n>
AArch64	TRCCLAIMCLR	Trace Claim Tag Clear Register
External	TRCCLAIMCLR	Trace Claim Tag Clear Register
AArch64	TRCCLAIMSET	Trace Claim Tag Set Register
External	TRCCLAIMSET	Trace Claim Tag Set Register
AArch64	TRCCNTCTLR<n>	Trace Counter Control Register <n>
External	TRCCNTCTLR<n>	Trace Counter Control Register <n>
AArch64	TRCCNTRLDVR<n>	Trace Counter Reload Value Register <n>
External	TRCCNTRLDVR<n>	Trace Counter Reload Value Register <n>
AArch64	TRCCNTVR<n>	Trace Counter Value Register <n>
External	TRCCNTVR<n>	Trace Counter Value Register <n>
AArch64	TRCCONFIGR	Trace Configuration Register
External	TRCCONFIGR	Trace Configuration Register
AArch64	TRCEVENTCTL0R	Trace Event Control 0 Register
External	TRCEVENTCTL0R	Trace Event Control 0 Register
AArch64	TRCEVENTCTL1R	Trace Event Control 1 Register
External	TRCEVENTCTL1R	Trace Event Control 1 Register
AArch64	TRCEXTINSELR<n>	Trace External Input Select Register <n>
External	TRCEXTINSELR<n>	Trace External Input Select Register <n>
AArch64	TRCIDR0	Trace ID Register 0
External	TRCIDR0	Trace ID Register 0
AArch64	TRCIDR1	Trace ID Register 1
External	TRCIDR1	Trace ID Register 1
AArch64	TRCIDR10	Trace ID Register 10
External	TRCIDR10	Trace ID Register 10
AArch64	TRCIDR11	Trace ID Register 11
External	TRCIDR11	Trace ID Register 11
AArch64	TRCIDR12	Trace ID Register 12
External	TRCIDR12	Trace ID Register 12
AArch64	TRCIDR13	Trace ID Register 13
External	TRCIDR13	Trace ID Register 13
AArch64	TRCIDR2	Trace ID Register 2
External	TRCIDR2	Trace ID Register 2
AArch64	TRCIDR3	Trace ID Register 3
External	TRCIDR3	Trace ID Register 3
AArch64	TRCIDR4	Trace ID Register 4
External	TRCIDR4	Trace ID Register 4
AArch64	TRCIDR5	Trace ID Register 5
External	TRCIDR5	Trace ID Register 5
AArch64	TRCIDR6	Trace ID Register 6
External	TRCIDR6	Trace ID Register 6
AArch64	TRCIDR7	Trace ID Register 7
External	TRCIDR7	Trace ID Register 7
AArch64	TRCIDR8	Trace ID Register 8
External	TRCIDR8	Trace ID Register 8
AArch64	TRCIDR9	Trace ID Register 9
External	TRCIDR9	Trace ID Register 9
AArch64	TRCIMSPEC0	Trace IMP DEF Register 0
External	TRCIMSPEC0	Trace IMP DEF Register 0
AArch64	TRCIMSPEC<n>	Trace IMP DEF Register <n>
External	TRCIMSPEC<n>	Trace IMP DEF Register <n>
AArch64	TRCITECR_EL1	Instrumentation Trace Control Register (EL1)
AArch64	TRCITECR_EL2	Instrumentation Trace Control Register (EL2)

Exec state	Name	Description
AArch64	TRCITEEDCR	Instrumentation Trace Extension External Debug Control Register
External	TRCITEEDCR	Instrumentation Trace Extension External Debug Control Register
AArch64	TRCPRGCTLR	Trace Programming Control Register
External	TRCPRGCTLR	Trace Programming Control Register
AArch64	TRCQCTLR	Trace Q Element Control Register
External	TRCQCTLR	Trace Q Element Control Register
AArch64	TRCRSCTLR<n>	Trace Resource Selection Control Register <n>
External	TRCRSCTLR<n>	Trace Resource Selection Control Register <n>
AArch64	TRCRSR	Trace Resources Status Register
External	TRCRSR	Trace Resources Status Register
AArch64	TRCSEQEVR<n>	Trace Sequencer State Transition Control Register <n>
External	TRCSEQEVR<n>	Trace Sequencer State Transition Control Register <n>
AArch64	TRCSEQRSTEVR	Trace Sequencer Reset Control Register
External	TRCSEQRSTEVR	Trace Sequencer Reset Control Register
AArch64	TRCSEQSTR	Trace Sequencer State Register
External	TRCSEQSTR	Trace Sequencer State Register
AArch64	TRCSSCCR<n>	Trace Single-shot Comparator Control Register <n>
External	TRCSSCCR<n>	Trace Single-shot Comparator Control Register <n>
AArch64	TRCSSCSR<n>	Trace Single-shot Comparator Control Status Register <n>
External	TRCSSCSR<n>	Trace Single-shot Comparator Control Status Register <n>
AArch64	TRCSSPCICR<n>	Trace Single-shot Processing Element Comparator Input Control Register <n>
External	TRCSSPCICR<n>	Trace Single-shot Processing Element Comparator Input Control Register <n>
AArch64	TRCSTALLCTLR	Trace Stall Control Register
External	TRCSTALLCTLR	Trace Stall Control Register
AArch64	TRCSTATR	Trace Status Register
External	TRCSTATR	Trace Status Register
AArch64	TRCSYNCPR	Trace Synchronization Period Register
External	TRCSYNCPR	Trace Synchronization Period Register
AArch64	TRCTRACEIDR	Trace ID Register
External	TRCTRACEIDR	Trace ID Register
AArch64	TRCTSCTLR	Trace Timestamp Control Register
External	TRCTSCTLR	Trace Timestamp Control Register
AArch64	TRCVICTLR	Trace ViewInst Main Control Register
External	TRCVICTLR	Trace ViewInst Main Control Register
AArch64	TRCVIIECTLR	Trace ViewInst Include/Exclude Control Register
External	TRCVIIECTLR	Trace ViewInst Include/Exclude Control Register
AArch64	TRCVIPCSSCTLR	Trace ViewInst Start/Stop PE Comparator Control Register
External	TRCVIPCSSCTLR	Trace ViewInst Start/Stop PE Comparator Control Register
AArch64	TRCVISSCTLR	Trace ViewInst Start/Stop Control Register
External	TRCVISSCTLR	Trace ViewInst Start/Stop Control Register
AArch64	TRCVMIDCCTLR0	Trace Virtual Context Identifier Comparator Control Register 0
External	TRCVMIDCCTLR0	Trace Virtual Context Identifier Comparator Control Register 0
AArch64	TRCVMIDCCTLR1	Trace Virtual Context Identifier Comparator Control Register 1
External	TRCVMIDCCTLR1	Trace Virtual Context Identifier Comparator Control Register 1
AArch64	TRCVMIDCVR<n>	Trace Virtual Context Identifier Comparator Value Register <n>
External	TRCVMIDCVR<n>	Trace Virtual Context Identifier Comparator Value Register <n>

In the CTI functional group:

Exec state	Name	Description
External	ASICCTL	CTI External Multiplexer Control register
External	CTIAPPCLEAR	CTI Application Trigger Clear register
External	CTIAPPPULSE	CTI Application Pulse register
External	CTIAPPSET	CTI Application Trigger Set register
External	CTIAUTHSTATUS	CTI Authentication Status register
External	CTICHINSTATUS	CTI Channel In Status register
External	CTICHOUTSTATUS	CTI Channel Out Status register
External	CTICIDR0	CTI Component Identification Register 0
External	CTICIDR1	CTI Component Identification Register 1
External	CTICIDR2	CTI Component Identification Register 2
External	CTICIDR3	CTI Component Identification Register 3
External	CTICLAIMCLR	CTI CLAIM Tag Clear register
External	CTICLAIMSET	CTI CLAIM Tag Set register

Exec state	Name	Description
External	CTICONTROL	CTI Control register
External	CTIDEVAFF0	CTI Device Affinity register 0
External	CTIDEVAFF1	CTI Device Affinity register 1
External	CTIDEVARCH	CTI Device Architecture register
External	CTIDEVCTL	CTI Device Control register
External	CTIDEVID	CTI Device ID register 0
External	CTIDEVID1	CTI Device ID register 1
External	CTIDEVID2	CTI Device ID register 2
External	CTIDEVTYPE	CTI Device Type register
External	CTIGATE	CTI Channel Gate Enable register
External	CTIINEN<n>	CTI Input Trigger to Output Channel Enable registers
External	CTIINTACK	CTI Output Trigger Acknowledge register
External	CTIITCTRL	CTI Integration mode Control register
External	CTILAR	CTI Lock Access Register
External	CTILSR	CTI Lock Status Register
External	CTIOUTEN<n>	CTI Input Channel to Output Trigger Enable registers
External	CTIPIDR0	CTI Peripheral Identification Register 0
External	CTIPIDR1	CTI Peripheral Identification Register 1
External	CTIPIDR2	CTI Peripheral Identification Register 2
External	CTIPIDR3	CTI Peripheral Identification Register 3
External	CTIPIDR4	CTI Peripheral Identification Register 4
External	CTITRIGINSTATUS	CTI Trigger In Status register
External	CTITRIGOUTSTATUS	CTI Trigger Out Status register

In the GICC functional group:

Exec state	Name	Description
External	GICC_ABPR	CPU Interface Aliased Binary Point Register
External	GICC_AEOIR	CPU Interface Aliased End Of Interrupt Register
External	GICC_AHPPIR	CPU Interface Aliased Highest Priority Pending Interrupt Register
External	GICC_AIAR	CPU Interface Aliased Interrupt Acknowledge Register
External	GICC_APR<n>	CPU Interface Active Priorities Registers
External	GICC_BPR	CPU Interface Binary Point Register
External	GICC_CTLR	CPU Interface Control Register
External	GICC_DIR	CPU Interface Deactivate Interrupt Register
External	GICC_EOIR	CPU Interface End Of Interrupt Register
External	GICC_HPPIR	CPU Interface Highest Priority Pending Interrupt Register
External	GICC_IAR	CPU Interface Interrupt Acknowledge Register
External	GICC_IIDR	CPU Interface Identification Register
External	GICC_NSAPR<n>	CPU Interface Non-secure Active Priorities Registers
External	GICC_PMR	CPU Interface Priority Mask Register
External	GICC_RPR	CPU Interface Running Priority Register
External	GICC_STATUSR	CPU Interface Status Register

In the GICD functional group:

Exec state	Name	Description
External	GICD_CLRSPI_NSR	Clear Non-secure SPI Pending Register
External	GICD_CLRSPI_SR	Clear Secure SPI Pending Register
External	GICD_CPENDSGIR<n>	SGI Clear-Pending Registers
External	GICD_CTLR	Distributor Control Register
External	GICD_ICACTIVER<n>	Interrupt Clear-Active Registers
External	GICD_ICACTIVER<n>E	Interrupt Clear-Active Registers (extended SPI range)
External	GICD_ICENABLER<n>	Interrupt Clear-Enable Registers
External	GICD_ICENABLER<n>E	Interrupt Clear-Enable Registers
External	GICD_ICFGR<n>	Interrupt Configuration Registers
External	GICD_ICFGR<n>E	Interrupt Configuration Registers (Extended SPI Range)
External	GICD_ICPENDR<n>	Interrupt Clear-Pending Registers
External	GICD_ICPENDR<n>E	Interrupt Clear-Pending Registers (extended SPI range)
External	GICD_IGROUPR<n>	Interrupt Group Registers
External	GICD_IGROUPR<n>E	Interrupt Group Registers (extended SPI range)
External	GICD_IGRPMODR<n>	Interrupt Group Modifier Registers

Exec state	Name	Description
External	GICD_IGRPMODR<n>E	Interrupt Group Modifier Registers (extended SPI range)
External	GICD_IIDR	Distributor Implementer Identification Register
External	GICD_INMIR<n>	Non-maskable Interrupt Registers, x = 0 to 31
External	GICD_INMIR<n>E	Non-maskable Interrupt Registers for Extended SPIs, x = 0 to 31
External	GICD_IPRIORITYR<n>	Interrupt Priority Registers
External	GICD_IPRIORITYR<n>E	Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.
External	GICD_IROUTER<n>	Interrupt Routing Registers
External	GICD_IROUTER<n>E	Interrupt Routing Registers (Extended SPI Range)
External	GICD_ISACTIVER<n>	Interrupt Set-Active Registers
External	GICD_ISACTIVER<n>E	Interrupt Set-Active Registers (extended SPI range)
External	GICD_ISENABLER<n>	Interrupt Set-Enable Registers
External	GICD_ISENABLER<n>E	Interrupt Set-Enable Registers
External	GICD_ISPENDR<n>	Interrupt Set-Pending Registers
External	GICD_ISPENDR<n>E	Interrupt Set-Pending Registers (extended SPI range)
External	GICD_ITARGETSR<n>	Interrupt Processor Targets Registers
External	GICD_NSACR<n>	Non-secure Access Control Registers
External	GICD_NSACR<n>E	Non-secure Access Control Registers
External	GICD_SETSPI_NSR	Set Non-secure SPI Pending Register
External	GICD_SETSPI_SR	Set Secure SPI Pending Register
External	GICD_SGIR	Software Generated Interrupt Register
External	GICD_SPENDSGIR<n>	SGI Set-Pending Registers
External	GICD_STATUSR	Error Reporting Status Register
External	GICD_TYPER	Interrupt Controller Type Register
External	GICD_TYPER2	Interrupt Controller Type Register 2
External	GICM_CLRSPI_NSR	Clear Non-secure SPI Pending Register
External	GICM_CLRSPI_SR	Clear Secure SPI Pending Register
External	GICM_IIDR	Distributor Implementer Identification Register
External	GICM_SETSPI_NSR	Set Non-secure SPI Pending Register
External	GICM_SETSPI_SR	Set Secure SPI Pending Register
External	GICM_TYPER	Distributor MSI Type Register

In the GICH functional group:

Exec state	Name	Description
External	GICH_APR<n>	Active Priorities Registers
External	GICH_EISR	End Interrupt Status Register
External	GICH_ELRSR	Empty List Register Status Register
External	GICH_HCR	Hypervisor Control Register
External	GICH_LR<n>	List Registers
External	GICH_MISR	Maintenance Interrupt Status Register
External	GICH_VMCR	Virtual Machine Control Register
External	GICH_VTR	Virtual Type Register

In the GICR functional group:

Exec state	Name	Description
External	GICR_CLRLPIR	Clear LPI Pending Register
External	GICR_CTLR	Redistributor Control Register
External	GICR_ICACTIVER0	Interrupt Clear-Active Register 0
External	GICR_ICACTIVER<n>E	Interrupt Clear-Active Registers
External	GICR_ICENABLER0	Interrupt Clear-Enable Register 0
External	GICR_ICENABLER<n>E	Interrupt Clear-Enable Registers
External	GICR_ICFGR0	Interrupt Configuration Register 0
External	GICR_ICFGR1	Interrupt Configuration Register 1
External	GICR_ICFGR<n>E	Interrupt configuration registers
External	GICR_ICPENDR0	Interrupt Clear-Pending Register 0
External	GICR_ICPENDR<n>E	Interrupt Clear-Pending Registers
External	GICR_IGROUPR0	Interrupt Group Register 0
External	GICR_IGROUPR<n>E	Interrupt Group Registers
External	GICR_IGRPMODR0	Interrupt Group Modifier Register 0
External	GICR_IGRPMODR<n>E	Interrupt Group Modifier Registers
External	GICR_IIDR	Redistributor Implementer Identification Register

Exec state	Name	Description
External	GICR_INMIR0	Non-maskable Interrupt Register 0
External	GICR_INMIR<n>E	Non-maskable Interrupt Registers for Extended PPIs, x = 1 to 2.
External	GICR_INVALLR	Redistributor Invalidate All Register
External	GICR_INVLPIR	Redistributor Invalidate LPI Register
External	GICR_IPRIORITYR<n>	Interrupt Priority Registers
External	GICR_IPRIORITYR<n>E	Interrupt Priority Registers (extended PPI range)
External	GICR_ISACTIVER0	Interrupt Set-Active Register 0
External	GICR_ISACTIVER<n>E	Interrupt Set-Active Registers
External	GICR_ISENABLER0	Interrupt Set-Enable Register 0
External	GICR_ISENABLER<n>E	Interrupt Set-Enable Registers
External	GICR_ISPENDR0	Interrupt Set-Pending Register 0
External	GICR_ISPENDR<n>E	Interrupt Set-Pending Registers
External	GICR_MPAMIDR	Report maximum PARTID and PMG Register
External	GICR_NSACR	Non-secure Access Control Register
External	GICR_PARTIDR	Set PARTID and PMG Register
External	GICR_PENDBASER	Redistributor LPI Pending Table Base Address Register
External	GICR_PROPBASER	Redistributor Properties Base Address Register
External	GICR_SETLPIR	Set LPI Pending Register
External	GICR_STATUSR	Error Reporting Status Register
External	GICR_SYNCR	Redistributor Synchronize Register
External	GICR_TYPER	Redistributor Type Register
External	GICR_VPENDBASER	Virtual Redistributor LPI Pending Table Base Address Register
External	GICR_VPROPBASER	Virtual Redistributor Properties Base Address Register
External	GICR_VSGIPENDR	Redistributor virtual SGI pending state register
External	GICR_VSGIR	Redistributor virtual SGI pending state request register
External	GICR_WAKER	Redistributor Wake Register

In the GICV functional group:

Exec state	Name	Description
External	GICV_ABPR	Virtual Machine Aliased Binary Point Register
External	GICV_AEOIR	Virtual Machine Aliased End Of Interrupt Register
External	GICV_AHPPIR	Virtual Machine Aliased Highest Priority Pending Interrupt Register
External	GICV_AIAR	Virtual Machine Aliased Interrupt Acknowledge Register
External	GICV_APR<n>	Virtual Machine Active Priorities Registers
External	GICV_BPR	Virtual Machine Binary Point Register
External	GICV_CTLR	Virtual Machine Control Register
External	GICV_DIR	Virtual Machine Deactivate Interrupt Register
External	GICV_EOIR	Virtual Machine End Of Interrupt Register
External	GICV_HPPIR	Virtual Machine Highest Priority Pending Interrupt Register
External	GICV_IAR	Virtual Machine Interrupt Acknowledge Register
External	GICV_IIDR	Virtual Machine CPU Interface Identification Register
External	GICV_PMR	Virtual Machine Priority Mask Register
External	GICV_RPR	Virtual Machine Running Priority Register
External	GICV_STATUSR	Virtual Machine Error Reporting Status Register

In the GITS functional group:

Exec state	Name	Description
External	GITS_BASER<n>	ITS Table Descriptors
External	GITS_CBASER	ITS Command Queue Descriptor
External	GITS_CREADR	ITS Read Register
External	GITS_CTLR	ITS Control Register
External	GITS_CWRITER	ITS Write Register
External	GITS_IIDR	ITS Identification Register
External	GITS_MPAMIDR	Report maximum PARTID and PMG Register
External	GITS_MPIDR	Report ITS's affinity.
External	GITS_PARTIDR	Set PARTID and PMG Register
External	GITS_SGIR	ITS SGI Register
External	GITS_STATUSR	ITS Error Reporting Status Register
External	GITS_TRANSLATER	ITS Translation Register
External	GITS_TYPER	ITS Type Register

Exec state	Name	Description
External	GITS_UMSIR	ITS Unmapped MSI register

In the AMU functional group:

Exec state	Name	Description
AArch32	AMCFGR	Activity Monitors Configuration Register
External	AMCFGR	Activity Monitors Configuration Register
AArch64	AMCFGR_EL0	Activity Monitors Configuration Register
AArch64	AMCG1IDR_EL0	Activity Monitors Counter Group 1 Identification Register
AArch32	AMCGCR	Activity Monitors Counter Group Configuration Register
External	AMCGCR	Activity Monitors Counter Group Configuration Register
AArch64	AMCGCR_EL0	Activity Monitors Counter Group Configuration Register
External	AMCIDR0	Activity Monitors Component Identification Register 0
External	AMCIDR1	Activity Monitors Component Identification Register 1
External	AMCIDR2	Activity Monitors Component Identification Register 2
External	AMCIDR3	Activity Monitors Component Identification Register 3
External	AMCNTEN	Activity Monitors Count Set and Clear Register
External	AMCNTENCLR	Activity Monitors Count Enable Clear Register
AArch32	AMCNTENCLR0	Activity Monitors Count Enable Clear Register 0
External	AMCNTENCLR0	Activity Monitors Count Enable Clear Register 0
AArch64	AMCNTENCLR0_EL0	Activity Monitors Count Enable Clear Register 0
AArch32	AMCNTENCLR1	Activity Monitors Count Enable Clear Register 1
External	AMCNTENCLR1	Activity Monitors Count Enable Clear Register 1
AArch64	AMCNTENCLR1_EL0	Activity Monitors Count Enable Clear Register 1
External	AMCNTENSET	Activity Monitors Count Enable Set Register
AArch32	AMCNTENSET0	Activity Monitors Count Enable Set Register 0
External	AMCNTENSET0	Activity Monitors Count Enable Set Register 0
AArch64	AMCNTENSET0_EL0	Activity Monitors Count Enable Set Register 0
AArch32	AMCNTENSET1	Activity Monitors Count Enable Set Register 1
External	AMCNTENSET1	Activity Monitors Count Enable Set Register 1
AArch64	AMCNTENSET1_EL0	Activity Monitors Count Enable Set Register 1
AArch32	AMCR	Activity Monitors Control Register
External	AMCR	Activity Monitors Control Register
AArch64	AMCR_EL0	Activity Monitors Control Register
External	AMDEVAFF	Activity Monitors Device Affinity Register
External	AMDEVAFF0	Activity Monitors Device Affinity Register 0
External	AMDEVAFF1	Activity Monitors Device Affinity Register 1
External	AMDEVARCH	Activity Monitors Device Architecture Register
External	AMDEVTYPE	Activity Monitors Device Type Register
AArch32	AMEVCNTR0<n>	Activity Monitors Event Counter Registers 0
External	AMEVCNTR0<n>	Activity Monitors Event Counter Registers 0
AArch64	AMEVCNTR0<n>_EL0	Activity Monitors Event Counter Registers 0
AArch32	AMEVCNTR1<n>	Activity Monitors Event Counter Registers 1
External	AMEVCNTR1<n>	Activity Monitors Event Counter Registers 1
AArch64	AMEVCNTR1<n>_EL0	Activity Monitors Event Counter Registers 1
AArch64	AMEVCNTVOFF0<n>_EL2	Activity Monitors Event Counter Virtual Offset Registers 0
AArch64	AMEVCNTVOFF1<n>_EL2	Activity Monitors Event Counter Virtual Offset Registers 1
AArch32	AMEVTYPER0<n>	Activity Monitors Event Type Registers 0
External	AMEVTYPER0<n>	Activity Monitors Event Type Registers 0
AArch64	AMEVTYPER0<n>_EL0	Activity Monitors Event Type Registers 0
AArch32	AMEVTYPER1<n>	Activity Monitors Event Type Registers 1
External	AMEVTYPER1<n>	Activity Monitors Event Type Registers 1
AArch64	AMEVTYPER1<n>_EL0	Activity Monitors Event Type Registers 1
External	AMIIDR	Activity Monitors Implementation Identification Register
External	AMPIDR0	Activity Monitors Peripheral Identification Register 0
External	AMPIDR1	Activity Monitors Peripheral Identification Register 1
External	AMPIDR2	Activity Monitors Peripheral Identification Register 2
External	AMPIDR3	Activity Monitors Peripheral Identification Register 3
External	AMPIDR4	Activity Monitors Peripheral Identification Register 4
External	AMROOTCR	Activity Monitors Root Control Register
External	AMSCR	Activity Monitors Secure Control Register
AArch32	AMUSERENR	Activity Monitors User Enable Register
AArch64	AMUSERENR_EL0	Activity Monitors User Enable Register

In the BRBE functional group:

Exec state	Name	Description
AArch64	BRBCR_EL1	Branch Record Buffer Control Register (EL1)
AArch64	BRBCR_EL2	Branch Record Buffer Control Register (EL2)
AArch64	BRBFCR_EL1	Branch Record Buffer Function Control Register
AArch64	BRBIDR0_EL1	Branch Record Buffer ID0 Register
AArch64	BRBINF<n>_EL1	Branch Record Buffer Information Register <n>
AArch64	BRBINFINJ_EL1	Branch Record Buffer Information Injection Register
AArch64	BRBSRC<n>_EL1	Branch Record Buffer Source Address Register <n>
AArch64	BRBSRCINJ_EL1	Branch Record Buffer Source Address Injection Register
AArch64	BRBTGT<n>_EL1	Branch Record Buffer Target Address Register <n>
AArch64	BRBTGTINJ_EL1	Branch Record Buffer Target Address Injection Register
AArch64	BRBTS_EL1	Branch Record Buffer Timestamp Register

In the Trace Management functional group:

Exec state	Name	Description
AArch64	TRCAUTHSTATUS	Trace Authentication Status Register
External	TRCAUTHSTATUS	Trace Authentication Status Register
External	TRCCIDR0	Trace Component Identification Register 0
External	TRCCIDR1	Trace Component Identification Register 1
External	TRCCIDR2	Trace Component Identification Register 2
External	TRCCIDR3	Trace Component Identification Register 3
External	TRCDEVAFF	Trace Device Affinity Register
AArch64	TRCDEVARCH	Trace Device Architecture Register
External	TRCDEVARCH	Trace Device Architecture Register
AArch64	TRCDEVID	Trace Device Configuration Register
External	TRCDEVID	Trace Device Configuration Register
External	TRCDEVID1	Trace Device Configuration Register 1
External	TRCDEVID2	Trace Device Configuration Register 2
External	TRCDEVTYPE	Trace Device Type Register
External	TRCITCTRL	Trace Integration Mode Control Register
External	TRCLAR	Trace Lock Access Register
External	TRCLSR	Trace Lock Status Register
AArch64	TRCOSLSR	Trace OS Lock Status Register
External	TRCOSLSR	Trace OS Lock Status Register
External	TRCPDCR	Trace PowerDown Control Register
External	TRCPDSR	Trace PowerDown Status Register
External	TRCPIDR0	Trace Peripheral Identification Register 0
External	TRCPIDR1	Trace Peripheral Identification Register 1
External	TRCPIDR2	Trace Peripheral Identification Register 2
External	TRCPIDR3	Trace Peripheral Identification Register 3
External	TRCPIDR4	Trace Peripheral Identification Register 4
External	TRCPIDR5	Trace Peripheral Identification Register 5
External	TRCPIDR6	Trace Peripheral Identification Register 6
External	TRCPIDR7	Trace Peripheral Identification Register 7

In the Guarded Control Stack registers functional group:

Exec state	Name	Description
AArch64	GCSCRE0_EL1	Guarded Control Stack Control Register (EL0)
AArch64	GCSCR_EL1	Guarded Control Stack Control Register (EL1)
AArch64	GCSCR_EL2	Guarded Control Stack Control Register (EL2)
AArch64	GCSCR_EL3	Guarded Control Stack Control Register (EL3)
AArch64	GCSPR_EL0	Guarded Control Stack Pointer Register (EL0)
AArch64	GCSPR_EL1	Guarded Control Stack Pointer Register (EL1)
AArch64	GCSPR_EL2	Guarded Control Stack Pointer Register (EL2)
AArch64	GCSPR_EL3	Guarded Control Stack Pointer Register (EL3)

In the MPAM functional group:

Exec state	Name	Description
AArch64	MPAM0_EL1	MPAM0 Register (EL1)
AArch64	MPAM1_EL1	MPAM1 Register (EL1)
AArch64	MPAM2_EL2	MPAM2 Register (EL2)
AArch64	MPAM3_EL3	MPAM3 Register (EL3)
AArch64	MPAMBW0_EL1	MPAM PE-side Maximum-bandwidth Control Register (EL0)
AArch64	MPAMBW1_EL1	MPAM PE-side Maximum-bandwidth Control Register (EL1)
AArch64	MPAMBW2_EL2	MPAM PE-side Maximum-bandwidth Control Register (EL2)
AArch64	MPAMBW3_EL3	MPAM PE-side Maximum-bandwidth Control Register (EL3)
AArch64	MPAMBWCAP_EL2	MPAM PE-side Maximum-bandwidth Limit Virtualization Register
AArch64	MPAMBWIDR_EL1	MPAM PE-side Bandwidth Controls ID Register
AArch64	MPAMBWSM_EL1	MPAM Streaming Mode Bandwidth Control Register (EL1)
External	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register
External	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register
External	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register
External	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register
External	MPAMCFG_DIS	MPAM Partition Configuration Disable Register
External	MPAMCFG_EN	MPAM Partition Configuration Enable Register
External	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register
External	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register
External	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register
External	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register
External	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register
External	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register
External	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register
External	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register
External	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register
External	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register
External	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register
External	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register
External	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register
External	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register
External	MPAMCFG_PRI	MPAM Priority Partition Configuration Register
External	MPAMF_AIDR	MPAM Architecture Identification Register
External	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register
External	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register
External	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register
External	MPAMF_ECR	MPAM Error Control Register
External	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register
External	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register
External	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register
External	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register
External	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register
External	MPAMF_ESR	MPAM Error Status Register
External	MPAMF_IDR	MPAM Features Identification Register
External	MPAMF_IIDR	MPAM Implementation Identification Register
External	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register
External	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register
External	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register
External	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register
External	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register
External	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register
External	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register
External	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register
External	MPAMF_SIDR	MPAM Features Secure Identification Register
AArch64	MPAMHCR_EL2	MPAM Hypervisor Control Register (EL2)
AArch64	MPAMSM_EL1	MPAM Streaming Mode Register
AArch64	MPAMVPM0_EL2	MPAM Virtual PARTID Mapping Register 0
AArch64	MPAMVPM1_EL2	MPAM Virtual PARTID Mapping Register 1
AArch64	MPAMVPM2_EL2	MPAM Virtual PARTID Mapping Register 2
AArch64	MPAMVPM3_EL2	MPAM Virtual PARTID Mapping Register 3
AArch64	MPAMVPM4_EL2	MPAM Virtual PARTID Mapping Register 4

Exec state	Name	Description
AArch64	MPAMVPM5_EL2	MPAM Virtual PARTID Mapping Register 5
AArch64	MPAMVPM6_EL2	MPAM Virtual PARTID Mapping Register 6
AArch64	MPAMVPM7_EL2	MPAM Virtual PARTID Mapping Register 7
AArch64	MPAMVPMV_EL2	MPAM Virtual Partition Mapping Valid Register
External	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register
External	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register
External	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register
External	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register
External	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register
External	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register
External	MSMON_CSU	MPAM Cache Storage Usage Monitor Register
External	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register
External	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register
External	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register
External	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register
External	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register
External	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register
External	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register
External	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register
External	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register
External	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register
External	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register
External	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register
External	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register

In the SPE functional group:

Exec state	Name	Description
AArch64	PMBIDR_EL1	Profiling Buffer ID Register
AArch64	PMBLIMITR_EL1	Profiling Buffer Limit Address Register
AArch64	PMBPTR_EL1	Profiling Buffer Write Pointer Register
AArch64	PMBSR_EL1	Profiling Buffer Status/syndrome Register (EL1)
AArch64	PMBSR_EL2	Profiling Buffer Syndrome Register (EL2)
AArch64	PMBSR_EL3	Profiling Buffer Syndrome Register (EL3)
AArch64	PMSCR_EL1	Statistical Profiling Control Register (EL1)
AArch64	PMSCR_EL2	Statistical Profiling Control Register (EL2)
AArch64	PMSDSFR_EL1	Sampling Data Source Filter Register
AArch64	PMSEVFR_EL1	Sampling Event Filter Register
AArch64	PMSFCR_EL1	Sampling Filter Control Register
AArch64	PMSICR_EL1	Sampling Interval Counter Register
AArch64	PMSIDR_EL1	Sampling Profiling ID Register
AArch64	PMSIRR_EL1	Sampling Interval Reload Register
AArch64	PMSLATFR_EL1	Sampling Latency Filter Register
AArch64	PMSNEVFR_EL1	Sampling Inverted Event Filter Register

In the TRBE functional group:

Exec state	Name	Description
AArch64	PMBMAR_EL1	Profiling Buffer Memory Attribute Register
External	TRBAUTHSTATUS	Authentication Status Register
AArch64	TRBBASER_EL1	Trace Buffer Base Address Register
External	TRBBASER_EL1	Trace Buffer Base Address Register
External	TRBCIDR0	Component Identification Register 0
External	TRBCIDR1	Component Identification Register 1
External	TRBCIDR2	Component Identification Register 2
External	TRBCIDR3	Component Identification Register 3
External	TRBCR	Trace Buffer Control Register
External	TRBDEVAFF	Device Affinity Register

Exec state	Name	Description
External	TRBDEVARCH	Trace Buffer Device Architecture Register
External	TRBDEVID	Device Configuration Register
External	TRBDEVID1	Device Configuration Register 1
External	TRBDEVID2	Device Configuration Register 2
External	TRBDEVTYPE	Device Type Register
AArch64	TRBIDR_EL1	Trace Buffer ID Register
External	TRBIDR_EL1	Trace Buffer ID Register
External	TRBITCTRL	Integration Mode Control Register
External	TRBLAR	Lock Access Register
AArch64	TRBLIMITR_EL1	Trace Buffer Limit Address Register
External	TRBLIMITR_EL1	Trace Buffer Limit Address Register
External	TRBLSR	Lock Status Register
AArch64	TRBMAR_EL1	Trace Buffer Memory Attribute Register
External	TRBMAR_EL1	Trace Buffer Memory Attribute Register
AArch64	TRBMPAM_EL1	Trace Buffer MPAM Configuration Register
External	TRBMPAM_EL1	Trace Buffer MPAM Configuration Register
External	TRBPIDR0	Peripheral Identification Register 0
External	TRBPIDR1	Peripheral Identification Register 1
External	TRBPIDR2	Peripheral Identification Register 2
External	TRBPIDR3	Peripheral Identification Register 3
External	TRBPIDR4	Peripheral Identification Register 4
External	TRBPIDR5	Peripheral Identification Register 5
External	TRBPIDR6	Peripheral Identification Register 6
External	TRBPIDR7	Peripheral Identification Register 7
AArch64	TRBPTR_EL1	Trace Buffer Write Pointer Register
External	TRBPTR_EL1	Trace Buffer Write Pointer Register
AArch64	TRBSR_EL1	Trace Buffer Status/syndrome Register (EL1)
External	TRBSR_EL1	Trace Buffer Status/syndrome Register
AArch64	TRBSR_EL2	Trace Buffer Syndrome Register (EL2)
AArch64	TRBSR_EL3	Trace Buffer Syndrome Register (EL3)
AArch64	TRBTRG_EL1	Trace Buffer Trigger Counter Register
External	TRBTRG_EL1	Trace Buffer Trigger Counter Register

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

External Registers

AMCFGR: Activity Monitors Configuration Register

AMCGCR: Activity Monitors Counter Group Configuration Register

AMCIDR0: Activity Monitors Component Identification Register 0

AMCIDR1: Activity Monitors Component Identification Register 1

AMCIDR2: Activity Monitors Component Identification Register 2

AMCIDR3: Activity Monitors Component Identification Register 3

AMCNTEN: Activity Monitors Count Set and Clear Register

AMCNTENCLR: Activity Monitors Count Enable Clear Register

AMCNTENCLR0: Activity Monitors Count Enable Clear Register 0

AMCNTENCLR1: Activity Monitors Count Enable Clear Register 1

AMCNTENSET: Activity Monitors Count Enable Set Register

AMCNTENSET0: Activity Monitors Count Enable Set Register 0

AMCNTENSET1: Activity Monitors Count Enable Set Register 1

AMCR: Activity Monitors Control Register

AMDEVAFF: Activity Monitors Device Affinity Register

AMDEVAFF0: Activity Monitors Device Affinity Register 0

AMDEVAFF1: Activity Monitors Device Affinity Register 1

AMDEVARCH: Activity Monitors Device Architecture Register

AMDEVTYPE: Activity Monitors Device Type Register

AMEVCNTR0<n>: Activity Monitors Event Counter Registers 0

AMEVCNTR1<n>: Activity Monitors Event Counter Registers 1

AMEVTYPER0<n>: Activity Monitors Event Type Registers 0

AMEVTYPER1<n>: Activity Monitors Event Type Registers 1

AMIIDR: Activity Monitors Implementation Identification Register

AMPIDR0: Activity Monitors Peripheral Identification Register 0

AMPIDR1: Activity Monitors Peripheral Identification Register 1

AMPIDR2: Activity Monitors Peripheral Identification Register 2

AMPIDR3: Activity Monitors Peripheral Identification Register 3

AMPIDR4: Activity Monitors Peripheral Identification Register 4

AMROOTCR: Activity Monitors Root Control Register

AMSCR: Activity Monitors Secure Control Register

ASICCTL: CTI External Multiplexer Control register

CNTACR<n>: Counter-timer Access Control Registers

CNTCR: Counter Control Register

CNTCV: Counter Count Value register

CNTEL0ACR: Counter-timer EL0 Access Control Register

CNTFID0: Counter Frequency ID

CNTFID<n>: Counter Frequency IDs, n > 0

CNTFRQ: Counter-timer Frequency

CNTID: Counter Identification Register

CNTNSAR: Counter-timer Non-secure Access Register

CNTPCT: Counter-timer Physical Count

CNTP_CTL: Counter-timer Physical Timer Control

CNTP_CVAL: Counter-timer Physical Timer CompareValue

CNTP_TVAL: Counter-timer Physical Timer TimerValue

CNTSCR: Counter Scale Register

CNTSR: Counter Status Register

CNTTIDR: Counter-timer Timer ID Register

CNTVCT: Counter-timer Virtual Count

CNTVOFF: Counter-timer Virtual Offset

CNTVOFF<n>: Counter-timer Virtual Offsets

CNTV_CTL: Counter-timer Virtual Timer Control

CNTV_CVAL: Counter-timer Virtual Timer CompareValue

CNTV_TVAL: Counter-timer Virtual Timer TimerValue

CTIAPPCLEAR: CTI Application Trigger Clear register

CTIAPPPULSE: CTI Application Pulse register

CTIAPPSET: CTI Application Trigger Set register

CTIAUTHSTATUS: CTI Authentication Status register

CTICHINSTATUS: CTI Channel In Status register

CTICHOUTSTATUS: CTI Channel Out Status register

CTICIDR0: CTI Component Identification Register 0

CTICIDR1: CTI Component Identification Register 1

CTICIDR2: CTI Component Identification Register 2

CTICIDR3: CTI Component Identification Register 3

CTICLAIMCLR: CTI CLAIM Tag Clear register

CTICLAIMSET: CTI CLAIM Tag Set register

CTICONTROL: CTI Control register

CTIDEVAFF0: CTI Device Affinity register 0

CTIDEVAFF1: CTI Device Affinity register 1

CTIDEVARCH: CTI Device Architecture register

CTIDEVCTL: CTI Device Control register

CTIDEVID: CTI Device ID register 0

CTIDEVID1: CTI Device ID register 1

CTIDEVID2: CTI Device ID register 2

CTIDEVTYPE: CTI Device Type register

CTIGATE: CTI Channel Gate Enable register

CTIINEN<n>: CTI Input Trigger to Output Channel Enable registers

CTIINTACK: CTI Output Trigger Acknowledge register

CTIITCTRL: CTI Integration mode Control register

CTILAR: CTI Lock Access Register

CTILSR: CTI Lock Status Register

CTIOUTEN<n>: CTI Input Channel to Output Trigger Enable registers

CTIPIDR0: CTI Peripheral Identification Register 0

CTIPIDR1: CTI Peripheral Identification Register 1

CTIPIDR2: CTI Peripheral Identification Register 2

CTIPIDR3: CTI Peripheral Identification Register 3

CTIPIDR4: CTI Peripheral Identification Register 4

CTITRIGINSTATUS: CTI Trigger In Status register

CTITRIGOUTSTATUS: CTI Trigger Out Status register

CounterID<n>: Counter ID registers

DBGAUTHSTATUS_EL1: Debug Authentication Status Register

DBGBCR<n>_EL1: Debug Breakpoint Control Registers

DBGBVR<n>_EL1: Debug Breakpoint Value Registers

DBGCLAIMCLR_EL1: Debug CLAIM Tag Clear Register

DBGCLAIMSET_EL1: Debug CLAIM Tag Set Register

DBGDTRRX_EL0: Debug Data Transfer Register, Receive

DBGDTRTX_EL0: Debug Data Transfer Register, Transmit

DBGWCR<n>_EL1: Debug Watchpoint Control Registers

DBGWVR<n>_EL1: Debug Watchpoint Value Registers

EDAA32PFR: External Debug Auxiliary Processor Feature Register

EDACR: External Debug Auxiliary Control Register

EDCIDR0: External Debug Component Identification Register 0

EDCIDR1: External Debug Component Identification Register 1

EDCIDR2: External Debug Component Identification Register 2

EDCIDR3: External Debug Component Identification Register 3

EDCIDSR: External Debug Context ID Sample Register

EDDEVAFF0: External Debug Device Affinity register 0

EDDEVAFF1: External Debug Device Affinity register 1

EDDEVARCH: External Debug Device Architecture Register

EDDEVID: External Debug Device ID register 0

EDDEVID1: External Debug Device ID Register 1

EDDEVID2: External Debug Device ID register 2

EDDEVTYPE: External Debug Device Type register

EDDFR: External Debug Feature Register

EDDFR1: External Debug Feature Register 1

EDDFR2: External Debug Feature Register 2

EDECCR: External Debug Exception Catch Control Register

EDECR: External Debug Execution Control Register

EDESR: External Debug Event Status Register

EDHSR: External Debug Halting Syndrome Register

EDITCTRL: External Debug Integration mode Control register

EDITR: External Debug Instruction Transfer Register

EDLAR: External Debug Lock Access Register

EDLSR: External Debug Lock Status Register

EDPCSR: External Debug Program Counter Sample Register

EDPFR: External Debug Processor Feature Register

EDPIDR0: External Debug Peripheral Identification Register 0

EDPIDR1: External Debug Peripheral Identification Register 1

EDPIDR2: External Debug Peripheral Identification Register 2

EDPIDR3: External Debug Peripheral Identification Register 3

EDPIDR4: External Debug Peripheral Identification Register 4

EDPRCR: External Debug Power/Reset Control Register

EDPRSR: External Debug Processor Status Register

EDRCR: External Debug Reserve Control Register

EDSCR: External Debug Status and Control Register

EDSCR2: External Debug Status and Control Register 2

EDVIDSR: External Debug Virtual Context Sample Register

EDWAR: External Debug Watchpoint Address Register

ERR<n>ADDR: Error Record <n> Address Register

ERR<n>CTLR: Error Record <n> Control Register

ERR<n>FR: Error Record <n> Feature Register

ERR<n>MISC0: Error Record <n> Miscellaneous Register 0

ERR<n>MISC1: Error Record <n> Miscellaneous Register 1

ERR<n>MISC2: Error Record <n> Miscellaneous Register 2

ERR<n>MISC3: Error Record <n> Miscellaneous Register 3

ERR<n>PFGCDN: Error Record <n> Pseudo-fault Generation Countdown Register

ERR<n>PFGCTL: Error Record <n> Pseudo-fault Generation Control Register

ERR<n>PFGF: Error Record <n> Pseudo-fault Generation Feature Register

ERR<n>STATUS: Error Record <n> Primary Status Register

ERRACR: Access Configuration Register

ERRCIDR0: Component Identification Register 0

ERRCIDR1: Component Identification Register 1

ERRCIDR2: Component Identification Register 2

ERRCIDR3: Component Identification Register 3

ERRCRICR0: Critical Error Interrupt Configuration Register 0

ERRCRICR1: Critical Error Interrupt Configuration Register 1

ERRCRICR2: Critical Error Interrupt Configuration Register 2

ERRDEVAFF: Device Affinity Register

ERRDEVARCH: Device Architecture Register

ERRDEVID: Device Configuration Register

ERRERICR0: Error Recovery Interrupt Configuration Register 0

ERRERICR1: Error Recovery Interrupt Configuration Register 1

ERRERICR2: Error Recovery Interrupt Configuration Register 2

ERRFHICR0: Fault Handling Interrupt Configuration Register 0

ERRFHICR1: Fault Handling Interrupt Configuration Register 1

ERRFHICR2: Fault Handling Interrupt Configuration Register 2

ERRGSR<m>: Error Group <n> Status Register

ERRIIDR: Implementation Identification Register

ERRIMPDEF<n>: IMPLEMENTATION DEFINED Register <n>

ERRIRQCR<n>: Generic Error Interrupt Configuration Register <n>

ERRIRQSR: Error Interrupt Status Register

ERRPIDR0: Peripheral Identification Register 0

ERRPIDR1: Peripheral Identification Register 1

ERRPIDR2: Peripheral Identification Register 2

ERRPIDR3: Peripheral Identification Register 3

ERRPIDR4: Peripheral Identification Register 4

GICC_ABPR: CPU Interface Aliased Binary Point Register

GICC_AEOIR: CPU Interface Aliased End Of Interrupt Register

GICC_AHPPIR: CPU Interface Aliased Highest Priority Pending Interrupt Register

GICC_AIAR: CPU Interface Aliased Interrupt Acknowledge Register

GICC_APR<n>: CPU Interface Active Priorities Registers

GICC_BPR: CPU Interface Binary Point Register

GICC_CTLR: CPU Interface Control Register

GICC_DIR: CPU Interface Deactivate Interrupt Register

GICC_EOIR: CPU Interface End Of Interrupt Register

GICC_HPPIR: CPU Interface Highest Priority Pending Interrupt Register

GICC_IAR: CPU Interface Interrupt Acknowledge Register

GICC_IIDR: CPU Interface Identification Register

GICC_NSAPR<n>: CPU Interface Non-secure Active Priorities Registers

GICC_PMR: CPU Interface Priority Mask Register

GICC_RPR: CPU Interface Running Priority Register

GICC_STATUSR: CPU Interface Status Register

GICD_CLRSPI_NSR: Clear Non-secure SPI Pending Register

GICD_CLRSPI_SR: Clear Secure SPI Pending Register

GICD_CPENDSGIR<n>: SGI Clear-Pending Registers

GICD_CTLR: Distributor Control Register

GICD_ICACTIVER<n>: Interrupt Clear-Active Registers

GICD_ICACTIVER<n>E: Interrupt Clear-Active Registers (extended SPI range)

GICD_ICENABLER<n>: Interrupt Clear-Enable Registers

GICD_ICENABLER<n>E: Interrupt Clear-Enable Registers

GICD_ICFGR<n>: Interrupt Configuration Registers

GICD_ICFGR<n>E: Interrupt Configuration Registers (Extended SPI Range)

GICD_ICPENDR<n>: Interrupt Clear-Pending Registers

GICD_ICPENDR<n>E: Interrupt Clear-Pending Registers (extended SPI range)

GICD_IGROUPR<n>: Interrupt Group Registers

GICD_IGROUPR<n>E: Interrupt Group Registers (extended SPI range)

GICD_IGRPMODR<n>: Interrupt Group Modifier Registers

GICD_IGRPMODR<n>E: Interrupt Group Modifier Registers (extended SPI range)

GICD_IIDR: Distributor Implementer Identification Register

GICD_INMIR<n>: Non-maskable Interrupt Registers, x = 0 to 31

GICD_INMIR<n>E: Non-maskable Interrupt Registers for Extended SPIs, x = 0 to 31

GICD_IPRIORITYR<n>: Interrupt Priority Registers

GICD_IPRIORITYR<n>E: Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.

GICD_IROUTER<n>: Interrupt Routing Registers

GICD_IROUTER<n>E: Interrupt Routing Registers (Extended SPI Range)

GICD_ISACTIVER<n>: Interrupt Set-Active Registers

GICD_ISACTIVER<n>E: Interrupt Set-Active Registers (extended SPI range)

GICD_ISENABLER<n>: Interrupt Set-Enable Registers

GICD_ISENABLER<n>E: Interrupt Set-Enable Registers

GICD_ISPENDR<n>: Interrupt Set-Pending Registers

GICD_ISPENDR<n>E: Interrupt Set-Pending Registers (extended SPI range)

GICD_ITARGETSR<n>: Interrupt Processor Targets Registers

GICD_NSACR<n>: Non-secure Access Control Registers

GICD_NSACR<n>E: Non-secure Access Control Registers

GICD_SETSPI_NSR: Set Non-secure SPI Pending Register

GICD_SETSPI_SR: Set Secure SPI Pending Register

GICD_SGIR: Software Generated Interrupt Register

GICD_SPENDSGIR<n>: SGI Set-Pending Registers

GICD_STATUSR: Error Reporting Status Register

GICD_TYPER: Interrupt Controller Type Register

GICD_TYPER2: Interrupt Controller Type Register 2

GICH_APR<n>: Active Priorities Registers

GICH_EISR: End Interrupt Status Register

GICH_ELRSR: Empty List Register Status Register

GICH_HCR: Hypervisor Control Register

GICH_LR<n>: List Registers

GICH_MISR: Maintenance Interrupt Status Register

GICH_VMCR: Virtual Machine Control Register

GICH_VTR: Virtual Type Register

GICM_CLRSPI_NSR: Clear Non-secure SPI Pending Register

GICM_CLRSPI_SR: Clear Secure SPI Pending Register

GICM_IIDR: Distributor Implementer Identification Register

GICM_SETSPI_NSR: Set Non-secure SPI Pending Register

GICM_SETSPI_SR: Set Secure SPI Pending Register

GICM_TYPER: Distributor MSI Type Register

GICR_CLRLPIR: Clear LPI Pending Register

GICR_CTLR: Redistributor Control Register

GICR_ICACTIVER0: Interrupt Clear-Active Register 0

GICR_ICACTIVER<n>E: Interrupt Clear-Active Registers

GICR_ICENABLER0: Interrupt Clear-Enable Register 0

GICR_ICENABLER<n>E: Interrupt Clear-Enable Registers

GICR_ICFGR0: Interrupt Configuration Register 0

GICR_ICFGR1: Interrupt Configuration Register 1

GICR_ICFGR<n>E: Interrupt configuration registers

GICR_ICPENDR0: Interrupt Clear-Pending Register 0

GICR_ICPENDR<n>E: Interrupt Clear-Pending Registers

GICR_IGROUPR0: Interrupt Group Register 0

GICR_IGROUPR<n>E: Interrupt Group Registers

GICR_IGRPMODR0: Interrupt Group Modifier Register 0

GICR_IGRPMODR<n>E: Interrupt Group Modifier Registers

GICR_IIDR: Redistributor Implementer Identification Register

GICR_INMIR0: Non-maskable Interrupt Register 0

GICR_INMIR<n>E: Non-maskable Interrupt Registers for Extended PPIs, x = 1 to 2.

GICR_INVALLR: Redistributor Invalidate All Register

GICR_INVLPIR: Redistributor Invalidate LPI Register

GICR_IPRIORITYR<n>: Interrupt Priority Registers

GICR_IPRIORITYR<n>E: Interrupt Priority Registers (extended PPI range)

GICR_ISACTIVER0: Interrupt Set-Active Register 0

GICR_ISACTIVER<n>E: Interrupt Set-Active Registers

GICR_ISENABLER0: Interrupt Set-Enable Register 0

GICR_ISENABLER<n>E: Interrupt Set-Enable Registers

GICR_ISPENDR0: Interrupt Set-Pending Register 0

GICR_ISPENDR<n>E: Interrupt Set-Pending Registers

GICR_MPAMIDR: Report maximum PARTID and PMG Register

GICR_NSACR: Non-secure Access Control Register

GICR_PARTIDR: Set PARTID and PMG Register

GICR_PENDBASER: Redistributor LPI Pending Table Base Address Register

GICR_PROPBASER: Redistributor Properties Base Address Register

GICR_SETLPIR: Set LPI Pending Register

GICR_STATUSR: Error Reporting Status Register

GICR_SYNCR: Redistributor Synchronize Register

GICR_TYPER: Redistributor Type Register

GICR_VPENDBASER: Virtual Redistributor LPI Pending Table Base Address Register

GICR_VPROPBASER: Virtual Redistributor Properties Base Address Register

GICR_VSGIPENDR: Redistributor virtual SGI pending state register

GICR_VSGIR: Redistributor virtual SGI pending state request register

GICR_WAKER: Redistributor Wake Register

GICV_ABPR: Virtual Machine Aliased Binary Point Register

GICV_AEOIR: Virtual Machine Aliased End Of Interrupt Register

GICV_AHPPIR: Virtual Machine Aliased Highest Priority Pending Interrupt Register

GICV_AIAR: Virtual Machine Aliased Interrupt Acknowledge Register

GICV_APR<n>: Virtual Machine Active Priorities Registers

GICV_BPR: Virtual Machine Binary Point Register

GICV_CTLR: Virtual Machine Control Register

GICV_DIR: Virtual Machine Deactivate Interrupt Register

GICV_EOIR: Virtual Machine End Of Interrupt Register

GICV_HPPIR: Virtual Machine Highest Priority Pending Interrupt Register

GICV_IAR: Virtual Machine Interrupt Acknowledge Register

GICV_IIDR: Virtual Machine CPU Interface Identification Register

GICV_PMR: Virtual Machine Priority Mask Register

GICV_RPR: Virtual Machine Running Priority Register

GICV_STATUSR: Virtual Machine Error Reporting Status Register

GITS_BASER<n>: ITS Table Descriptors

GITS_CBASER: ITS Command Queue Descriptor

GITS_CREADR: ITS Read Register

GITS_CTLR: ITS Control Register

GITS_CWRITER: ITS Write Register

GITS_IIDR: ITS Identification Register

GITS_MPAMIDR: Report maximum PARTID and PMG Register

GITS_MPIDR: Report ITS's affinity.

GITS_PARTIDR: Set PARTID and PMG Register

GITS_SGIR: ITS SGI Register

GITS_STATUSR: ITS Error Reporting Status Register

GITS_TRANSLATER: ITS Translation Register

GITS_TYPER: ITS Type Register

GITS_UMSIR: ITS Unmapped MSI register

MIDR_EL1: Main ID Register

MPAMCFG_CASSOC: MPAM Cache Maximum Associativity Partition Configuration Register

MPAMCFG_CMAX: MPAM Cache Maximum Capacity Partition Configuration Register

MPAMCFG_CMIN: MPAM Cache Minimum Capacity Partition Configuration Register

MPAMCFG_CPBM<n>: MPAM Cache Portion Bitmap Partition Configuration Register

MPAMCFG_DIS: MPAM Partition Configuration Disable Register

MPAMCFG_EN: MPAM Partition Configuration Enable Register

MPAMCFG_EN_FLAGS: MPAM Partition Configuration Enable Flags Register

MPAMCFG_INTPARTID: MPAM Internal PARTID Narrowing Configuration Register

MPAMCFG_IN_TL: MPAM Ingress PARTID Translation Configuration Register

MPAMCFG_IN_TL_BASE: MPAM Ingress PARTID Translation Base Configuration Register

MPAMCFG_IN_TL_MASK: MPAM Ingress PARTID Translation Mask Configuration Register

MPAMCFG_MBW_MAX: MPAM Memory Bandwidth Maximum Partition Configuration Register

MPAMCFG_MBW_MIN: MPAM Memory Bandwidth Minimum Partition Configuration Register

MPAMCFG_MBW_PBM<n>: MPAM Bandwidth Portion Bitmap Partition Configuration Register

MPAMCFG_MBW_PROP: MPAM Memory Bandwidth Proportional Stride Partition Configuration Register

MPAMCFG_MBW_WINWD: MPAM Memory Bandwidth Partitioning Window Width Configuration Register

MPAMCFG_OUT_TL: MPAM Egress PARTID Translation Configuration Register

MPAMCFG_OUT_TL_BASE: MPAM Egress PARTID Translation Base Configuration Register

MPAMCFG_OUT_TL_MASK: MPAM Egress PARTID Translation Mask Configuration Register

MPAMCFG_PART_SEL: MPAM Partition Configuration Selection Register

MPAMCFG_PRI: MPAM Priority Partition Configuration Register

MPAMF_AIDR: MPAM Architecture Identification Register

MPAMF_CCAP_IDR: MPAM Features Cache Capacity Partitioning ID register

MPAMF_CPOR_IDR: MPAM Features Cache Portion Partitioning ID register

MPAMF_CSUMON_IDR: MPAM Features Cache Storage Usage Monitoring ID register

MPAMF_ECR: MPAM Error Control Register

MPAMF_ERR_MSI_ADDR_H: MPAM Error MSI High-part Address Register

MPAMF_ERR_MSI_ADDR_L: MPAM Error MSI Low-part Address Register

MPAMF_ERR_MSI_ATTR: MPAM Error MSI Write Attributes Register

MPAMF_ERR_MSI_DATA: MPAM Error MSI Data Register

MPAMF_ERR_MSI_MPAM: MPAM Error MSI Write MPAM Information Register

MPAMF_ESR: MPAM Error Status Register

MPAMF_IDR: MPAM Features Identification Register

MPAMF_IIDR: MPAM Implementation Identification Register

MPAMF_IMPL_IDR: MPAM Implementation-Specific Partitioning Feature Identification Register

MPAMF_IN_TL_IDR: MPAM Ingress PARTID Translation ID Register

MPAMF_MBWUMON_IDR: MPAM Features Memory Bandwidth Usage Monitoring ID register

MPAMF_MBW_IDR: MPAM Memory Bandwidth Partitioning Identification Register

MPAMF_MSMON_IDR: MPAM Resource Monitoring Identification Register

MPAMF_OUT_TL_IDR: MPAM Egress PARTID Translation ID Register

MPAMF_PARTID_NRW_IDR: MPAM PARTID Narrowing ID register

MPAMF_PRI_IDR: MPAM Priority Partitioning Identification Register

MPAMF_SIDR: MPAM Features Secure Identification Register

MSMON_CAPT_EVNT: MPAM Capture Event Generation Register

MSMON_CFG_CSU_CTL: MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register

MSMON_CFG_CSU_FLT: MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register

MSMON_CFG_MBWU_CTL: MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register

MSMON_CFG_MBWU_FLT: MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register

MSMON_CFG_MON_SEL: MPAM Monitor Instance Selection Register

MSMON_CSU: MPAM Cache Storage Usage Monitor Register

MSMON_CSU_CAPTURE: MPAM Cache Storage Usage Monitor Capture Register

MSMON_CSU_OFSR: MPAM CSU Monitor Overflow Status Register

MSMON_MBWU: MPAM Memory Bandwidth Usage Monitor Register

MSMON_MBWU_CAPTURE: MPAM Memory Bandwidth Usage Monitor Capture Register

MSMON_MBWU_L: MPAM Long Memory Bandwidth Usage Monitor Register

MSMON_MBWU_L_CAPTURE: MPAM Long Memory Bandwidth Usage Monitor Capture Register

MSMON_MBWU_OFSR: MPAM MBWU Monitor Overflow Status Register

MSMON_OFLOW_MSI_ADDR_H: MPAM Monitor Overflow MSI Write High-part Address Register

MSMON_OFLOW_MSI_ADDR_L: MPAM Monitor Overflow MSI Low-part Address Register

MSMON_OFLOW_MSI_ATTR: MPAM Monitor Overflow MSI Write Attributes Register

MSMON_OFLOW_MSI_DATA: MPAM Monitor Overflow MSI Write Data Register

MSMON_OFLOW_MSI_MPAM: MPAM Monitor Overflow MSI Write MPAM Information Register

MSMON_OFLOW_SR: MPAM Monitor Overflow Status Register

OSLAR_EL1: OS Lock Access Register

PMAUTHSTATUS: Performance Monitors Authentication Status register

PMCCFILTR_EL0: Performance Monitors Cycle Counter Filter Register

PMCCIDSR: CONTEXTIDR_ELx Sample Register

PMCCNTR_EL0: Performance Monitors Cycle Counter

PMCCNTSVR_EL1: Performance Monitors Cycle Count Saved Value Register

PMCCR: PMU Configuration Control Register

PMCEID0: Performance Monitors Common Event Identification register 0

PMCEID1: Performance Monitors Common Event Identification register 1

PMCEID2: Performance Monitors Common Event Identification register 2

PMCEID3: Performance Monitors Common Event Identification register 3

PMCFGR: Performance Monitors Configuration Register

PMCGCR0: Counter Group Configuration Register 0

PMCID1SR: CONTEXTIDR_EL1 Sample Register

PMCID2SR: CONTEXTIDR_EL2 Sample Register

PMCIDR0: Performance Monitors Component Identification Register 0

PMCIDR1: Performance Monitors Component Identification Register 1

PMCIDR2: Performance Monitors Component Identification Register 2

PMCIDR3: Performance Monitors Component Identification Register 3

PMCNTEN: Performance Monitors Count Enable register

PMCNTENCLR_EL0: Performance Monitors Count Enable Clear Register

PMCNTENSET_EL0: Performance Monitors Count Enable Set Register

PMCR_EL0: Performance Monitors Control Register

PMDEVAFF: Performance Monitors Device Affinity register

PMDEVAFF0: Performance Monitors Device Affinity register 0

PMDEVAFF1: Performance Monitors Device Affinity register 1

PMDEVARCH: Performance Monitors Device Architecture register

PMDEVID: Performance Monitors Device ID register

PMDEVTYPE: Performance Monitors Device Type register

PMEVCNTR<n>_EL0: Performance Monitors Event Count Registers

PMEVCNTSVR<n>_EL1: Performance Monitors Event Count Saved Value Registers

PMEVFILT2R<n>: Performance Monitors Event Filter Registers

PMEVTYPER<n>_EL0: Performance Monitors Event Type Registers

PMICFILTR_EL0: Performance Monitors Instruction Counter Filter Register

PMICNTR_EL0: Performance Monitors Instruction Counter Register

PMICNTSVR_EL1: Performance Monitors Instruction Count Saved Value Register

PMIIDR: Performance Monitors Implementation Identification Register

PMINTEN: Performance Monitors Interrupt Enable register

PMINTENCLR_EL1: Performance Monitors Interrupt Enable Clear Register

PMINTENSET_EL1: Performance Monitors Interrupt Enable Set Register

PMITCTRL: Performance Monitors Integration mode Control register

PMLAR: Performance Monitors Lock Access Register

PMLSR: Performance Monitors Lock Status Register

PMMIR: Performance Monitors Machine Identification Register

PMOVS: Performance Monitors Overflow Flag Status register

PMOVSCLR_EL0: Performance Monitors Overflow Flag Status Clear register

PMOVSSET_EL0: Performance Monitors Overflow Flag Status Set Register

PMPCSCTL: PC Sample-based Profiling Control Register

PMPCSR: Program Counter Sample Register

PMPIDR0: Performance Monitors Peripheral Identification Register 0

PMPIDR1: Performance Monitors Peripheral Identification Register 1

PMPIDR2: Performance Monitors Peripheral Identification Register 2

PMPIDR3: Performance Monitors Peripheral Identification Register 3

PMPIDR4: Performance Monitors Peripheral Identification Register 4

PMSSCR_EL1: Performance Monitors Snapshot Status and Capture Register

PMSWINC_EL0: Performance Monitors Software Increment Register

PMVCIDSR: CONTEXTIDR_EL1 and VMID Sample Register

PMVIDSR: VMID Sample Register

PMZR_EL0: Performance Monitors Zero with Mask

TRBAUTHSTATUS: Authentication Status Register

TRBBASER_EL1: Trace Buffer Base Address Register

TRBCIDR0: Component Identification Register 0

TRBCIDR1: Component Identification Register 1

TRBCIDR2: Component Identification Register 2

TRBCIDR3: Component Identification Register 3

TRBCR: Trace Buffer Control Register

TRBDEVAFF: Device Affinity Register

TRBDEVARCH: Trace Buffer Device Architecture Register

TRBDEVID: Device Configuration Register

TRBDEVID1: Device Configuration Register 1

TRBDEVID2: Device Configuration Register 2

TRBDEVTYPE: Device Type Register

TRBIDR_EL1: Trace Buffer ID Register

TRBITCTRL: Integration Mode Control Register

TRBLAR: Lock Access Register

TRBLIMITR_EL1: Trace Buffer Limit Address Register

TRBLSR: Lock Status Register

TRBMAR_EL1: Trace Buffer Memory Attribute Register

TRBMPAM_EL1: Trace Buffer MPAM Configuration Register

TRBPIDR0: Peripheral Identification Register 0

TRBPIDR1: Peripheral Identification Register 1

TRBPIDR2: Peripheral Identification Register 2

TRBPIDR3: Peripheral Identification Register 3

TRBPIDR4: Peripheral Identification Register 4

TRBPIDR5: Peripheral Identification Register 5

TRBPIDR6: Peripheral Identification Register 6

TRBPIDR7: Peripheral Identification Register 7

TRBPTR_EL1: Trace Buffer Write Pointer Register

TRBSR_EL1: Trace Buffer Status/syndrome Register

TRBTRG_EL1: Trace Buffer Trigger Counter Register

TRCACATR<n>: Trace Address Comparator Access Type Register <n>

TRCACVR<n>: Trace Address Comparator Value Register <n>

TRCAUTHSTATUS: Trace Authentication Status Register

TRCAUXCTLR: Trace Auxiliary Control Register

TRCBBCTLR: Trace Branch Broadcast Control Register

TRCCCCTLR: Trace Cycle Count Control Register

TRCCIDCCTLR0: Trace Context Identifier Comparator Control Register 0

TRCCIDCCTLR1: Trace Context Identifier Comparator Control Register 1

TRCCIDCVR<n>: Trace Context Identifier Comparator Value Registers <n>

TRCCIDR0: Trace Component Identification Register 0

TRCCIDR1: Trace Component Identification Register 1

TRCCIDR2: Trace Component Identification Register 2

TRCCIDR3: Trace Component Identification Register 3

TRCCLAIMCLR: Trace Claim Tag Clear Register

TRCCLAIMSET: Trace Claim Tag Set Register

TRCCNTCTLR<n>: Trace Counter Control Register <n>

TRCCNTRLDVR<n>: Trace Counter Reload Value Register <n>

TRCCNTVR<n>: Trace Counter Value Register <n>

TRCCONFIGR: Trace Configuration Register

TRCDEVAFF: Trace Device Affinity Register

TRCDEVARCH: Trace Device Architecture Register

TRCDEVID: Trace Device Configuration Register

TRCDEVID1: Trace Device Configuration Register 1

TRCDEVID2: Trace Device Configuration Register 2

TRCDEVTYPE: Trace Device Type Register

TRCEVENTCTL0R: Trace Event Control 0 Register

TRCEVENTCTL1R: Trace Event Control 1 Register

TRCEXTINSELR<n>: Trace External Input Select Register <n>

TRCIDR0: Trace ID Register 0

TRCIDR1: Trace ID Register 1

TRCIDR10: Trace ID Register 10

TRCIDR11: Trace ID Register 11

TRCIDR12: Trace ID Register 12

TRCIDR13: Trace ID Register 13

TRCIDR2: Trace ID Register 2

TRCIDR3: Trace ID Register 3

TRCIDR4: Trace ID Register 4

TRCIDR5: Trace ID Register 5

TRCIDR6: Trace ID Register 6

TRCIDR7: Trace ID Register 7

TRCIDR8: Trace ID Register 8

TRCIDR9: Trace ID Register 9

TRCIMSPEC0: Trace IMP DEF Register 0

TRCIMSPEC<n>: Trace IMP DEF Register <n>

TRCITCTRL: Trace Integration Mode Control Register

TRCITEEDCR: Instrumentation Trace Extension External Debug Control Register

TRCLAR: Trace Lock Access Register

TRCLSR: Trace Lock Status Register

TRCOSLSR: Trace OS Lock Status Register

TRCPDCR: Trace PowerDown Control Register

TRCPDSR: Trace PowerDown Status Register

TRCPIDR0: Trace Peripheral Identification Register 0

TRCPIDR1: Trace Peripheral Identification Register 1

TRCPIDR2: Trace Peripheral Identification Register 2

TRCPIDR3: Trace Peripheral Identification Register 3

TRCPIDR4: Trace Peripheral Identification Register 4

TRCPIDR5: Trace Peripheral Identification Register 5

TRCPIDR6: Trace Peripheral Identification Register 6

TRCPIDR7: Trace Peripheral Identification Register 7

TRCPRGCTLR: Trace Programming Control Register

TRCQCTLR: Trace Q Element Control Register

TRCRSCTLR<n>: Trace Resource Selection Control Register <n>

TRCRSR: Trace Resources Status Register

TRCSEQEVR<n>: Trace Sequencer State Transition Control Register <n>

TRCSEQRSTEVR: Trace Sequencer Reset Control Register

TRCSEQSTR: Trace Sequencer State Register

TRCSSCCR<n>: Trace Single-shot Comparator Control Register <n>

TRCSSCSR<n>: Trace Single-shot Comparator Control Status Register <n>

TRCSSPCICR<n>: Trace Single-shot Processing Element Comparator Input Control Register <n>

TRCSTALLCTLR: Trace Stall Control Register

TRCSTATR: Trace Status Register

TRCSYNCPR: Trace Synchronization Period Register

TRCTRACEIDR: Trace ID Register

TRCTSCTLR: Trace Timestamp Control Register

TRCVICTLR: Trace ViewInst Main Control Register

TRCVIIECTLR: Trace ViewInst Include/Exclude Control Register

TRCVIPCSSCTLR: Trace ViewInst Start/Stop PE Comparator Control Register

TRCVISSCTLR: Trace ViewInst Start/Stop Control Register

TRCVMIDCCTLR0: Trace Virtual Context Identifier Comparator Control Register 0

TRCVMIDCCTLR1: Trace Virtual Context Identifier Comparator Control Register 1

TRCVMIDCVR<n>: Trace Virtual Context Identifier Comparator Value Register <n>

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

2025-06_rel

External register index by offset

Below are indexes for external registers in the following blocks:

In the AMU block:

Offset	Name	Description	Access	Accessor Condition	R
0x000 + (8 * n) for n in 16:0	AMEVCNTR0<n>	Activity Monitors Event Counter Registers 0	RO	When FEAT_AMU_EXT64 is implemented	Wh is i
0x000 + (8 * n) for n in 16:0	AMEVCNTR0<n>	Activity Monitors Event Counter Registers 0	RO	When FEAT_AMU_EXT32 is implemented	Wh is i
0x100 + (8 * n) for n in 16:0	AMEVCNTR1<n>	Activity Monitors Event Counter Registers 1	RO	When FEAT_AMU_EXT64 is implemented	Wh is i
0x100 + (8 * n) for n in 16:0	AMEVCNTR1<n>	Activity Monitors Event Counter Registers 1	RO	When FEAT_AMU_EXT32 is implemented	Wh is i
0x400 + (8 * n) for n in 16:0	AMEVTYPER0<n>	Activity Monitors Event Type Registers 0	RO	When FEAT_AMU_EXT64 is implemented	Wh is i
0x400 + (4 * n) for n in 16:0	AMEVTYPER0<n>	Activity Monitors Event Type Registers 0	RO	When FEAT_AMU_EXT32 is implemented	Wh is i
0x480 + (4 * n) for n in 16:0	AMEVTYPER1<n>	Activity Monitors Event Type Registers 1	RO	When FEAT_AMU_EXT32 is implemented	Wh is i
0x500 + (8 * n) for n in 16:0	AMEVTYPER1<n>	Activity Monitors Event Type Registers 1	RO	When FEAT_AMU_EXT64 is implemented	Wh is i
0xC00	AMCNTENSET	Activity Monitors Count Enable Set Register	RO	When FEAT_AMU_EXT64 is implemented	Wh is i FE imp
0xC00	AMCNTENSET0	Activity Monitors	RO	When FEAT_AMU_EXT32 is implemented	Wh is i

Offset	Name	Description	Access	Accessor Condition	R
		Count Enable Set Register 0			FE imp
0xC04	AMCNTENSET1	Activity Monitors Count Enable Set Register 1	RO	When FEAT_AMU_EXT32 is implemented	Wh is i FE imp
0xC10	AMCNTEN	Activity Monitors Count Set and Clear Register	RO	When FEAT_AMU_EXT64 is implemented	Wh is i FE imp
0xC20	AMCNTENCLR	Activity Monitors Count Enable Clear Register	RO	When FEAT_AMU_EXT64 is implemented	Wh is i FE imp
0xC20	AMCNTENCLR0	Activity Monitors Count Enable Clear Register 0	RO	When FEAT_AMU_EXT32 is implemented	Wh is i FE imp
0xC24	AMCNTENCLR1	Activity Monitors Count Enable Clear Register 1	RO	When FEAT_AMU_EXT32 is implemented	Wh is i
0xCE0	AMCGCR	Activity Monitors Counter Group Configuration Register	RO	When FEAT_AMU_EXT64 is implemented	Wh is i
0xCE0	AMCGCR	Activity Monitors Counter Group Configuration Register	RO	When FEAT_AMU_EXT32 is implemented	Wh is i
0xE00	AMCFGR	Activity Monitors Configuration Register	RO	When FEAT_AMU_EXT64 is implemented	Wh is i
0xE00	AMCFGR	Activity Monitors Configuration Register	RO	When FEAT_AMU_EXT32 is implemented	Wh is i
0xE04	AMCR	Activity Monitors Control Register	RO	When FEAT_AMU_EXT32 is implemented	Wh is i
0xE08	AMIIDR	Activity Monitors Implementation Identification Register	RO	When FEAT_AMU_EXT64 is implemented	Wh is i
0xE08	AMIIDR	Activity Monitors Implementation Identification Register	RO	When FEAT_AMU_EXT32 is implemented	Wh is i
0xE10	AMCR	Activity Monitors Control Register	RO	When FEAT_AMU_EXT64 is implemented	Wh is i
0xE40	AMSCR	Activity Monitors Secure Control Register	RW	When FEAT_AMU_EXTACR is implemented and FEAT_RME is not implemented	Wh FE is i FE imp

Offset	Name	Description	Access	Accessor Condition	R
0xE48	AMROOTCR	Activity Monitors Root Control Register	RW	When FEAT_AMU_EXTACR is implemented and FEAT_RME is implemented	Wh FEA is ir FEA imp
0xFA8	AMDEVAFF	Activity Monitors Device Affinity Register	RO	When FEAT_AMU_EXT64 is implemented	Wh is ir FEA imp imp imp AM
0xFA8	AMDEVAFF0	Activity Monitors Device Affinity Register 0	RO	When FEAT_AMU_EXT32 is implemented	Wh is ir FEA imp imp imp AM
0xFAC	AMDEVAFF1	Activity Monitors Device Affinity Register 1	RO	When FEAT_AMU_EXT32 is implemented	Wh is ir FEA imp imp imp AM
0xFBC	AMDEVARCH	Activity Monitors Device Architecture Register	RO	When FEAT_AMU_EXT64 is implemented	Wh is ir imp imp AM
0xFBC	AMDEVARCH	Activity Monitors Device Architecture Register	RO	When FEAT_AMU_EXT32 is implemented	Wh is ir imp imp AM
0xFCC	AMDEVTYPE	Activity Monitors Device Type Register	RO	When FEAT_AMU_EXT64 is implemented	Wh is ir imp imp AM
0xFCC	AMDEVTYPE	Activity Monitors Device Type Register	RO	When FEAT_AMU_EXT32 is implemented	Wh is ir imp imp AM
0xFD0	AMPIDR4	Activity Monitors Peripheral Identification Register 4	RO	When FEAT_AMUv1 is implemented	Wh is ir imp imp
0xFE0	AMPIDR0	Activity Monitors Peripheral Identification Register 0	RO	When FEAT_AMUv1 is implemented	Wh is ir imp imp
0xFE4	AMPIDR1	Activity Monitors Peripheral Identification Register 1	RO	When FEAT_AMUv1 is implemented	Wh is ir imp imp
0xFE8	AMPIDR2	Activity Monitors Peripheral Identification Register 2	RO	When FEAT_AMUv1 is implemented	Wh is ir imp imp

Offset	Name	Description	Access	Accessor Condition	R
0xFEC	AMPIDR3	Activity Monitors Peripheral Identification Register 3	RO	When FEAT_AMUv1 is implemented	Wh is i imp imp
0xFF0	AMCIDR0	Activity Monitors Component Identification Register 0	RO	When FEAT_AMUv1 is implemented	Wh is i imp imp
0xFF4	AMCIDR1	Activity Monitors Component Identification Register 1	RO	When FEAT_AMUv1 is implemented	Wh is i imp imp
0xFF8	AMCIDR2	Activity Monitors Component Identification Register 2	RO	When FEAT_AMUv1 is implemented	Wh is i imp imp
0xFFC	AMCIDR3	Activity Monitors Component Identification Register 3	RO	When FEAT_AMUv1 is implemented	Wh is i imp imp

In the CTI block:

Offset	Name	Description	Access	
0x000	CTICONTROL	CTI Control register	RW	-
0x010	CTIINTACK	CTI Output Trigger Acknowledge register	WO	-
0x014	CTIAPPSET	CTI Application Trigger Set register	RW	-
0x018	CTIAPPCLEAR	CTI Application Trigger Clear register	WO	-
0x01C	CTIAPPPULSE	CTI Application Pulse register	WO	-
0x020 + (4 * n)	CTIINEN<n>	CTI Input Trigger to Output Channel Enable registers	RW	-
0x0A0 + (4 * n)	CTIOUTEN<n>	CTI Input Channel to Output Trigger Enable registers	RW	-
0x130	CTITRIGINSTATUS	CTI Trigger In Status register	RO	-
0x134	CTITRIGOUTSTATUS	CTI Trigger Out Status register	RO	-
0x138	CTICHINSTATUS	CTI Channel In Status register	RO	-
0x13C	CTICHOUTSTATUS	CTI Channel Out Status register	RO	-
0x140	CTIGATE	CTI Channel Gate Enable register	RW	-
0x144	ASICCTL	CTI External Multiplexer Control register	RO	-
0x150	CTIDEVCTL	CTI Device Control register	RW	-
0xF00	CTIITCTRL	CTI Integration mode Control register	RW	-
0xFA0	CTICLAIMSET	CTI CLAIM Tag Set register	RW	-
0xFA4	CTICLAIMCLR	CTI CLAIM Tag Clear register	RW	-
0xFA8	CTIDEVAFF0	CTI Device Affinity register 0	RO	-
0xFAC	CTIDEVAFF1	CTI Device Affinity register 1	RO	-
0xFB0	CTILAR	CTI Lock Access Register	WO	-
0xFB4	CTILSR	CTI Lock Status Register	RO	-
0xFB8	CTIAUTHSTATUS	CTI Authentication Status register	RO	-
0xFBC	CTIDEVARCH	CTI Device Architecture register	RO	-
0xFC0	CTIDEVID2	CTI Device ID register 2	RO	-
0xFC4	CTIDEVID1	CTI Device ID register 1	RO	-
0xFC8	CTIDEVID	CTI Device ID register 0	RO	-

Offset	Name	Description	Access	
0xFCC	CTIDEVTYPE	CTI Device Type register	RO	-
0xFD0	CTIPIDR4	CTI Peripheral Identification Register 4	RO	-
0xFE0	CTIPIDR0	CTI Peripheral Identification Register 0	RO	-
0xFE4	CTIPIDR1	CTI Peripheral Identification Register 1	RO	-
0xFE8	CTIPIDR2	CTI Peripheral Identification Register 2	RO	-
0xFEC	CTIPIDR3	CTI Peripheral Identification Register 3	RO	-
0xFF0	CTICIDR0	CTI Component Identification Register 0	RO	-
0xFF4	CTICIDR1	CTI Component Identification Register 1	RO	-
0xFF8	CTICIDR2	CTI Component Identification Register 2	RO	-
0xFFC	CTICIDR3	CTI Component Identification Register 3	RO	-

In the Debug block:

Offset	Name	Description	Access	
0x020	EDESR	External Debug Event Status Register	RW	-
0x024	EDECR	External Debug Execution Control Register	RW	-
0x028	EDSCR2	External Debug Status and Control Register 2	RW	-
0x030	EDWAR	External Debug Watchpoint Address Register	RO	-
0x038	EDHSR	External Debug Halting Syndrome Register	RO	-
0x080	DBGDTRRX_EL0	Debug Data Transfer Register, Receive	RW	-
0x084	EDITR	External Debug Instruction Transfer Register	WO	-
0x088	EDSCR	External Debug Status and Control Register	RW	-
0x08C	DBGDTRTX_EL0	Debug Data Transfer Register, Transmit	RW	-
0x090	EDRCR	External Debug Reserve Control Register	WO	-
0x094	EDACR	External Debug Auxiliary Control Register	RW	-
0x098	EDECCR	External Debug Exception Catch Control Register	RW	-
0x0A0	EDPCSR[31:0]	External Debug Program Counter Sample Register	RO	-
0x0A0	EDPCSR[31:0]	External Debug Program Counter Sample Register	RO	-
0x0A4	EDCIDSR	External Debug Context ID Sample Register	RO	-
0x0A8	EDVIDSR	External Debug Virtual Context Sample Register	RO	-
0x0AC	EDPCSR[63:32]	External Debug Program Counter Sample Register	RO	-
0x0AC	EDPCSR[63:32]	External Debug Program Counter Sample Register	RO	-
0x300	OSLAR_EL1	OS Lock Access Register	WO	-
0x310	EDPRCR	External Debug Power/Reset Control Register	RW	-
0x314	EDPRSR	External Debug Processor Status Register	RO	-
0x400 + (16 * n)	DBGBVR<n>_EL1[63:0]	Debug Breakpoint Value Registers	RW	-
0x408 + (16 * n)	DBGBCR<n>_EL1	Debug Breakpoint Control Registers	RW	-
0x800 + (16 * n)	DBGWVR<n>_EL1[63:0]	Debug Watchpoint Value Registers	RW	-
0x808 + (16 * n)	DBGWCR<n>_EL1	Debug Watchpoint Control Registers	RW	-
0xD00	MIDR_EL1	Main ID Register	RO	-
0xD20	EDPFR	External Debug Processor Feature Register	RO	-
0xD28	EDDFR	External Debug Feature Register	RO	-
0xD48	EDDFR1	External Debug Feature Register 1	RO	-
0xD50	EDDFR2	External Debug Feature Register 2	RO	-
0xD60	EDAA32PFR	External Debug Auxiliary Processor Feature Register	RO	-
0xF00	EDITCTRL	External Debug Integration mode Control register	RW	-
0xFA0	DBGCLAIMSET_EL1	Debug CLAIM Tag Set Register	RW	-
0xFA4	DBGCLAIMCLR_EL1	Debug CLAIM Tag Clear Register	RW	-
0xFA8	EDDEVAFF0	External Debug Device Affinity register 0	RO	-

Offset	Name	Description	Access	
0xFAC	EDDEVAFF1	External Debug Device Affinity register 1	RO	-
0xFB0	EDLAR	External Debug Lock Access Register	WO	-
0xFB4	EDLSR	External Debug Lock Status Register	RO	-
0xFB8	DBGAUTHSTATUS_EL1	Debug Authentication Status Register	RO	-
0xFBC	EDDEVARCH	External Debug Device Architecture Register	RO	-
0xFC0	EDDEVID2	External Debug Device ID register 2	RO	-
0xFC4	EDDEVID1	External Debug Device ID Register 1	RO	-
0xFC8	EDDEVID	External Debug Device ID register 0	RO	-
0xFCC	EDDEVTYPE	External Debug Device Type register	RO	-
0xFD0	EDPIDR4	External Debug Peripheral Identification Register 4	RO	-
0xFE0	EDPIDR0	External Debug Peripheral Identification Register 0	RO	-
0xFE4	EDPIDR1	External Debug Peripheral Identification Register 1	RO	-
0xFE8	EDPIDR2	External Debug Peripheral Identification Register 2	RO	-
0xFEC	EDPIDR3	External Debug Peripheral Identification Register 3	RO	-
0xFF0	EDCIDR0	External Debug Component Identification Register 0	RO	-
0xFF4	EDCIDR1	External Debug Component Identification Register 1	RO	-
0xFF8	EDCIDR2	External Debug Component Identification Register 2	RO	-
0xFFC	EDCIDR3	External Debug Component Identification Register 3	RO	-

In the ETE block:

Offset	Name	Description	Access	
0x004	TRCPRGCTLR	Trace Programming Control Register	RW	-
0x00C	TRCSTATR	Trace Status Register	RO	-
0x010	TRCCONFIGR	Trace Configuration Register	RW	-
0x018	TRCAUXCTLR	Trace Auxiliary Control Register	RW	-
0x020	TRCEVENTCTL0R	Trace Event Control 0 Register	RW	-
0x024	TRCEVENTCTL1R	Trace Event Control 1 Register	RW	-
0x028	TRCRSR	Trace Resources Status Register	RW	-
0x02C	TRCSTALLCTLR	Trace Stall Control Register	RW	-
0x030	TRCTSCTLR	Trace Timestamp Control Register	RW	-
0x034	TRCSYNCPR	Trace Synchronization Period Register	RW	-
0x038	TRCCCCTLR	Trace Cycle Count Control Register	RW	-
0x03C	TRCBBCTLR	Trace Branch Broadcast Control Register	RW	-
0x040	TRCTRACEIDR	Trace ID Register	RW	-
0x044	TRCQCTLR	Trace Q Element Control Register	RW	-
0x048	TRCITEEDCR	Instrumentation Trace Extension External Debug Control Register	RW	-
0x080	TRCVICTLR	Trace ViewInst Main Control Register	RW	-
0x084	TRCVIIECTLR	Trace ViewInst Include/Exclude Control Register	RW	-
0x088	TRCVISSCTLR	Trace ViewInst Start/Stop Control Register	RW	-
0x08C	TRCVIPCSSCTLR	Trace ViewInst Start/Stop PE Comparator Control Register	RW	-
0x100 + (4 * n)	TRCSEQEVR<n>	Trace Sequencer State Transition Control Register <n>	RW	-
0x118	TRCSEQRSTEVR	Trace Sequencer Reset Control Register	RW	-
0x11C	TRCSEQSTR	Trace Sequencer State Register	RW	-
0x120 + (4 * n)	TRCEXTINSELR<n>	Trace External Input Select Register <n>	RW	-
0x140 + (4 * n)	TRCCNTRLDVR<n>	Trace Counter Reload Value Register <n>	RW	-
0x150 + (4 * n)	TRCCNTCTLR<n>	Trace Counter Control Register <n>	RW	-
0x160 + (4 * n)	TRCCNTVR<n>	Trace Counter Value Register <n>	RW	-

Offset	Name	Description	Access	
0x180	TRCIDR8	Trace ID Register 8	RO	-
0x184	TRCIDR9	Trace ID Register 9	RO	-
0x188	TRCIDR10	Trace ID Register 10	RO	-
0x18C	TRCIDR11	Trace ID Register 11	RO	-
0x190	TRCIDR12	Trace ID Register 12	RO	-
0x194	TRCIDR13	Trace ID Register 13	RO	-
0x1C0	TRCIMSPEC0	Trace IMP DEF Register 0	RW	-
0x1C0 + (4 * n)	TRCIMSPEC<n>	Trace IMP DEF Register <n>	RW	-
0x1E0	TRCIDR0	Trace ID Register 0	RO	-
0x1E4	TRCIDR1	Trace ID Register 1	RO	-
0x1E8	TRCIDR2	Trace ID Register 2	RO	-
0x1EC	TRCIDR3	Trace ID Register 3	RO	-
0x1F0	TRCIDR4	Trace ID Register 4	RO	-
0x1F4	TRCIDR5	Trace ID Register 5	RO	-
0x1F8	TRCIDR6	Trace ID Register 6	RO	-
0x1FC	TRCIDR7	Trace ID Register 7	RO	-
0x200 + (4 * n)	TRCRSCTLR<n>	Trace Resource Selection Control Register <n>	RW	-
0x280 + (4 * n)	TRCSSCCR<n>	Trace Single-shot Comparator Control Register <n>	RW	-
0x2A0 + (4 * n)	TRCSSCSR<n>	Trace Single-shot Comparator Control Status Register <n>	RW	-
0x2C0 + (4 * n)	TRCSSPCICR<n>	Trace Single-shot Processing Element Comparator Input Control Register <n>	RW	-
0x304	TRCOSLSR	Trace OS Lock Status Register	RO	-
0x310	TRCPDCR	Trace PowerDown Control Register	RW	-
0x314	TRCPDSR	Trace PowerDown Status Register	RO	-
0x400 + (8 * n)	TRCACVR<n>	Trace Address Comparator Value Register <n>	RW	-
0x480 + (8 * n)	TRCACATR<n>	Trace Address Comparator Access Type Register <n>	RW	-
0x600 + (8 * n)	TRCCIDCVR<n>	Trace Context Identifier Comparator Value Registers <n>	RW	-
0x640 + (8 * n)	TRCVMIDCVR<n>	Trace Virtual Context Identifier Comparator Value Register <n>	RW	-
0x680	TRCCIDCCTLR0	Trace Context Identifier Comparator Control Register 0	RW	-
0x684	TRCCIDCCTLR1	Trace Context Identifier Comparator Control Register 1	RW	-
0x688	TRCVMIDCCTLR0	Trace Virtual Context Identifier Comparator Control Register 0	RW	-
0x68C	TRCVMIDCCTLR1	Trace Virtual Context Identifier Comparator Control Register 1	RW	-
0xF00	TRCITCTRL	Trace Integration Mode Control Register	RW	-
0xFA0	TRCCLAIMSET	Trace Claim Tag Set Register	RW	-
0xFA4	TRCCLAIMCLR	Trace Claim Tag Clear Register	RW	-
0xFA8	TRCDEVAFF	Trace Device Affinity Register	RO	-
0xFB0	TRCLAR	Trace Lock Access Register	WO	-
0xFB4	TRCLSR	Trace Lock Status Register	RO	-
0xFB8	TRCAUTHSTATUS	Trace Authentication Status Register	RO	-
0xFBC	TRCDEVARCH	Trace Device Architecture Register	RO	-
0xFC0	TRCDEVID2	Trace Device Configuration Register 2	RO	-
0xFC4	TRCDEVID1	Trace Device Configuration Register 1	RO	-
0xFC8	TRCDEVID	Trace Device Configuration Register	RO	-
0xFCC	TRCDEVTYPE	Trace Device Type Register	RO	-
0xFD0	TRCPIDR4	Trace Peripheral Identification Register 4	RO	-
0xFD4	TRCPIDR5	Trace Peripheral Identification Register 5	RO	-
0xFD8	TRCPIDR6	Trace Peripheral Identification Register 6	RO	-
0xFDC	TRCPIDR7	Trace Peripheral Identification Register 7	RO	-
0xFE0	TRCPIDR0	Trace Peripheral Identification Register 0	RO	-

Offset	Name	Description	Access	
0xFE4	TRCPIDR1	Trace Peripheral Identification Register 1	RO	-
0xFE8	TRCPIDR2	Trace Peripheral Identification Register 2	RO	-
0xFEC	TRCPIDR3	Trace Peripheral Identification Register 3	RO	-
0xFF0	TRCCIDR0	Trace Component Identification Register 0	RO	-
0xFF4	TRCCIDR1	Trace Component Identification Register 1	RO	-
0xFF8	TRCCIDR2	Trace Component Identification Register 2	RO	-
0xFFC	TRCCIDR3	Trace Component Identification Register 3	RO	-

In the GIC CPU interface block:

Offset	Name	Description	Access	
0x0000	GICC_CTLR	CPU Interface Control Register	RW	-
0x0004	GICC_PMR	CPU Interface Priority Mask Register	RW	-
0x0008	GICC_BPR	CPU Interface Binary Point Register	RW	-
0x000C	GICC_IAR	CPU Interface Interrupt Acknowledge Register	RO	-
0x0010	GICC_EOIR	CPU Interface End Of Interrupt Register	WO	-
0x0014	GICC_RPR	CPU Interface Running Priority Register	RO	-
0x0018	GICC_HPPIR	CPU Interface Highest Priority Pending Interrupt Register	RO	-
0x001C	GICC_ABPR	CPU Interface Aliased Binary Point Register	RW	-
0x0020	GICC_AIAR	CPU Interface Aliased Interrupt Acknowledge Register	RO	-
0x0024	GICC_AEOIR	CPU Interface Aliased End Of Interrupt Register	WO	-
0x0028	GICC_AHPPIR	CPU Interface Aliased Highest Priority Pending Interrupt Register	RO	-
0x002C	GICC_STATUSR	CPU Interface Status Register	RW	-
0x002C	GICC_STATUSR	CPU Interface Status Register	RW	-
0x002C	GICC_STATUSR	CPU Interface Status Register	RW	-
0x002C	GICC_STATUSR	CPU Interface Status Register	RW	-
0x00D0 + (4 * n)	GICC_APR<n>	CPU Interface Active Priorities Registers	RW	-
0x00E0 + (4 * n)	GICC_NSAPR<n>	CPU Interface Non-secure Active Priorities Registers	RW	-
0x00FC	GICC_IIDR	CPU Interface Identification Register	RO	-
0x1000	GICC_DIR	CPU Interface Deactivate Interrupt Register	WO	-

In the GIC Distributor block:

In the Dist_base block:

Offset	Name	Description	Access
0x0000	GICD_CTLR	Distributor Control Register	RW
0x0004	GICD_TYPER	Interrupt Controller Type Register	RO
0x0008	GICD_IIDR	Distributor Implementer Identification Register	RO
0x000C	GICD_TYPER2	Interrupt Controller Type Register 2	RO
0x0010	GICD_STATUSR	Error Reporting Status Register	RW
0x0010	GICD_STATUSR	Error Reporting Status Register	RW
0x0010	GICD_STATUSR	Error Reporting Status Register	RW
0x0010	GICD_STATUSR	Error Reporting Status Register	RW
0x0040	GICD_SETSPI_NSR	Set Non-secure SPI Pending Register	WO
0x0048	GICD_CLRSPI_NSR	Clear Non-secure SPI Pending Register	WO
0x0050	GICD_SETSPI_SR	Set Secure SPI Pending Register	WO
0x0058	GICD_CLRSPI_SR	Clear Secure SPI Pending Register	WO
0x0080 + (4 * n)	GICD_IGROUPR<n>	Interrupt Group Registers	RW

Offset	Name	Description	Access
0x0100 + (4 * n)	GICD_ISENABLER<n>	Interrupt Set-Enable Registers	RW
0x0180 + (4 * n)	GICD_ICENABLER<n>	Interrupt Clear-Enable Registers	RW
0x0200 + (4 * n)	GICD_ISPENDR<n>	Interrupt Set-Pending Registers	RW
0x0280 + (4 * n)	GICD_ICPENDR<n>	Interrupt Clear-Pending Registers	RW
0x0300 + (4 * n)	GICD_ISACTIVER<n>	Interrupt Set-Active Registers	RW
0x0380 + (4 * n)	GICD_ICACTIVER<n>	Interrupt Clear-Active Registers	RW
0x0400 + (4 * n)	GICD_IPRIORITYR<n>	Interrupt Priority Registers	RW
0x0800 + (4 * n)	GICD_ITARGETSR<n>	Interrupt Processor Targets Registers	RW
0x0C00 + (4 * n)	GICD_ICFGR<n>	Interrupt Configuration Registers	RW
0x0D00 + (4 * n)	GICD_IGRPMODR<n>	Interrupt Group Modifier Registers	RW
0x0E00 + (4 * n)	GICD_NSACR<n>	Non-secure Access Control Registers	RW
0x0F00	GICD_SGIR	Software Generated Interrupt Register	WO
0x0F10 + (4 * n)	GICD_CPENDSGIR<n>	SGI Clear-Pending Registers	RW
0x0F20 + (4 * n)	GICD_SPENDSGIR<n>	SGI Set-Pending Registers	RW
0x0F80 + (4 * n)	GICD_INMIR<n>	Non-maskable Interrupt Registers, x = 0 to 31	RW
0x1000 + (4 * n)	GICD_IGROUPR<n>E	Interrupt Group Registers (extended SPI range)	RW
0x1200 + (4 * n)	GICD_ISENABLER<n>E	Interrupt Set-Enable Registers	RW
0x1400 + (4 * n)	GICD_ICENABLER<n>E	Interrupt Clear-Enable Registers	RW
0x1600 + (4 * n)	GICD_ISPENDR<n>E	Interrupt Set-Pending Registers (extended SPI range)	RW
0x1800 + (4 * n)	GICD_ICPENDR<n>E	Interrupt Clear-Pending Registers (extended SPI range)	RW
0x1A00 + (4 * n)	GICD_ISACTIVER<n>E	Interrupt Set-Active Registers (extended SPI range)	RW
0x1C00 + (4 * n)	GICD_ICACTIVER<n>E	Interrupt Clear-Active Registers (extended SPI range)	RW
0x2000 + (4 * n)	GICD_IPRIORITYR<n>E	Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.	RW
0x3000 + (4 * n)	GICD_ICFGR<n>E	Interrupt Configuration Registers (Extended SPI Range)	RW
0x3400 + (4 * n)	GICD_IGRPMODR<n>E	Interrupt Group Modifier Registers (extended SPI range)	RW
0x3600 + (4 * n)	GICD_NSACR<n>E	Non-secure Access Control Registers	RW
0x3B00 + (4 * n)	GICD_INMIR<n>E	Non-maskable Interrupt Registers for Extended SPIs, x = 0 to 31	RW
0x6000 + (8 * n)	GICD_IROUTER<n>	Interrupt Routing Registers	RW
0x8000 + (8 * n)	GICD_IROUTER<n>E	Interrupt Routing Registers (Extended SPI Range)	RW

In the MSI_base block:

Offset	Name	Description	Access
0x0004	GICM_TYPER	Distributor MSI Type Register	RO
0x0040	GICM_SETSPI_NSR	Set Non-secure SPI Pending Register	WO
0x0048	GICM_CLRSPI_NSR	Clear Non-secure SPI Pending Register	WO
0x0050	GICM_SETSPI_SR	Set Secure SPI Pending Register	WO
0x0058	GICM_CLRSPI_SR	Clear Secure SPI Pending Register	WO
0x0FCC	GICM_IIDR	Distributor Implementer Identification Register	RO

In the GIC ITS control block:

Offset	Name	Description	Access	
0x0000	GITS_CTLR	ITS Control Register	RW	-
0x0004	GITS_IIDR	ITS Identification Register	RO	-
0x0008	GITS_TYPER	ITS Type Register	RO	-
0x0010	GITS_MPAMIDR	Report maximum PARTID and PMG Register	RO	-
0x0014	GITS_PARTIDR	Set PARTID and PMG Register	RW	-
0x0018	GITS_MPIDR	Report ITS's affinity.	RO	-

Offset	Name	Description	Access	
0x0040	GITS_STATUSR	ITS Error Reporting Status Register	RW	-
0x0048	GITS_UMSIR	ITS Unmapped MSI register	RO	-
0x0080	GITS_CBASER	ITS Command Queue Descriptor	RW	-
0x0088	GITS_CWRITER	ITS Write Register	RW	-
0x0090	GITS_CREADR	ITS Read Register	RO	-
0x0100 + (8 * n)	GITS_BASER<n>	ITS Table Descriptors	RW	-
0x20020	GITS_SGIR	ITS SGI Register	WO	-

In the GIC ITS translation block:

Offset	Name	Description	Access	
0x0040	GITS_TRANSLATER	ITS Translation Register	WO	-

In the GIC Redistributor block:

In the RD_base block:

Offset	Name	Description	Access
0x0000	GICR_CTLR	Redistributor Control Register	RW
0x0004	GICR_IIDR	Redistributor Implementer Identification Register	RO
0x0008	GICR_TYPER	Redistributor Type Register	RO
0x0010	GICR_STATUSR	Error Reporting Status Register	RW
0x0010	GICR_STATUSR	Error Reporting Status Register	RW
0x0010	GICR_STATUSR	Error Reporting Status Register	RW
0x0010	GICR_STATUSR	Error Reporting Status Register	RW
0x0014	GICR_WAKER	Redistributor Wake Register	RW
0x0018	GICR_MPAMIDR	Report maximum PARTID and PMG Register	RO
0x001C	GICR_PARTIDR	Set PARTID and PMG Register	RW
0x0040	GICR_SETLPIR	Set LPI Pending Register	WO
0x0048	GICR_CLRLPIR	Clear LPI Pending Register	WO
0x0070	GICR_PROPBASER	Redistributor Properties Base Address Register	RW
0x0078	GICR_PENDBASER	Redistributor LPI Pending Table Base Address Register	RW
0x00A0	GICR_INVLPIR	Redistributor Invalidate LPI Register	WO
0x00B0	GICR_INVALLR	Redistributor Invalidate All Register	WO
0x00C0	GICR_SYNCR	Redistributor Synchronize Register	RO

In the SGI_base block:

Offset	Name	Description	Access
0x0080	GICR_IGROUPR0	Interrupt Group Register 0	RW
0x0080 + (4 * n)	GICR_IGROUPR<n>E	Interrupt Group Registers	RW
0x0100	GICR_ISENABLER0	Interrupt Set-Enable Register 0	RW
0x0100 + (4 * n)	GICR_ISENABLER<n>E	Interrupt Set-Enable Registers	RW
0x0180	GICR_ICENABLER0	Interrupt Clear-Enable Register 0	RW
0x0180 + (4 * n)	GICR_ICENABLER<n>E	Interrupt Clear-Enable Registers	RW
0x0200	GICR_ISPENDR0	Interrupt Set-Pending Register 0	RW
0x0200 + (4 * n)	GICR_ISPENDR<n>E	Interrupt Set-Pending Registers	RW
0x0280	GICR_ICPENDR0	Interrupt Clear-Pending Register 0	RW
0x0280 + (4 * n)	GICR_ICPENDR<n>E	Interrupt Clear-Pending Registers	RW

Offset	Name	Description	Access
0x0300	GICR_ISACTIVER0	Interrupt Set-Active Register 0	RW
0x0300 + (4 * n)	GICR_ISACTIVER<n>E	Interrupt Set-Active Registers	RW
0x0380	GICR_ICACTIVER0	Interrupt Clear-Active Register 0	RW
0x0380 + (4 * n)	GICR_ICACTIVER<n>E	Interrupt Clear-Active Registers	RW
0x0400 + (4 * n)	GICR_IPRIORITYR<n>	Interrupt Priority Registers	RW
0x0400 + (4 * n)	GICR_IPRIORITYR<n>E	Interrupt Priority Registers (extended PPI range)	RW
0x0C00	GICR_ICFGR0	Interrupt Configuration Register 0	RW
0x0C00 + (4 * n)	GICR_ICFGR<n>E	Interrupt configuration registers	RW
0x0C04	GICR_ICFGR1	Interrupt Configuration Register 1	RW
0x0D00	GICR_IGRPMODR0	Interrupt Group Modifier Register 0	RW
0x0D00 + (4 * n)	GICR_IGRPMODR<n>E	Interrupt Group Modifier Registers	RW
0x0E00	GICR_NSACR	Non-secure Access Control Register	RW
0x0F80	GICR_INMIR0	Non-maskable Interrupt Register 0	RW
0x0F80 + (4 * n)	GICR_INMIR<n>E	Non-maskable Interrupt Registers for Extended PPIs, x = 1 to 2.	RW

In the VLPI_base block:

Offset	Name	Description	Access
0x0070	GICR_VPROPBASER	Virtual Redistributor Properties Base Address Register	RW
0x0078	GICR_VPENDBASER	Virtual Redistributor LPI Pending Table Base Address Register	RW
0x0080	GICR_VSGIR	Redistributor virtual SGI pending state request register	WO
0x0088	GICR_VSGIPENDR	Redistributor virtual SGI pending state register	RO

In the GIC Virtual CPU interface block:

Offset	Name	Description	Access	
0x0000	GICV_CTLR	Virtual Machine Control Register	RW	-
0x0004	GICV_PMR	Virtual Machine Priority Mask Register	RW	-
0x0008	GICV_BPR	Virtual Machine Binary Point Register	RW	-
0x000C	GICV_IAR	Virtual Machine Interrupt Acknowledge Register	RO	-
0x0010	GICV_EOIR	Virtual Machine End Of Interrupt Register	WO	-
0x0014	GICV_RPR	Virtual Machine Running Priority Register	RO	-
0x0018	GICV_HPPIR	Virtual Machine Highest Priority Pending Interrupt Register	RO	-
0x001C	GICV_ABPR	Virtual Machine Aliased Binary Point Register	RW	-
0x0020	GICV_AIAR	Virtual Machine Aliased Interrupt Acknowledge Register	RO	-
0x0024	GICV_AEOIR	Virtual Machine Aliased End Of Interrupt Register	WO	-
0x0028	GICV_AHPPIR	Virtual Machine Aliased Highest Priority Pending Interrupt Register	RO	-
0x002C	GICV_STATUSR	Virtual Machine Error Reporting Status Register	RW	-
0x00D0 + (4 * n)	GICV_APR<n>	Virtual Machine Active Priorities Registers	RW	-
0x00FC	GICV_IIDR	Virtual Machine CPU Interface Identification Register	RO	-
0x1000	GICV_DIR	Virtual Machine Deactivate Interrupt Register	WO	-

In the GIC Virtual interface control block:

Offset	Name	Description	Access	
0x0000	GICH_HCR	Hypervisor Control Register	RW	-
0x0004	GICH_VTR	Virtual Type Register	RO	-
0x0008	GICH_VMCR	Virtual Machine Control Register	RW	-
0x0010	GICH_MISR	Maintenance Interrupt Status Register	RO	-

Offset	Name	Description	Access	
0x0020	GICH_EISR	End Interrupt Status Register	RO	-
0x0030	GICH_ELRSR	Empty List Register Status Register	RO	-
0x00F0 + (4 * n)	GICH_APR<n>	Active Priorities Registers	RW	-
0x0100 + (4 * n)	GICH_LR<n>	List Registers	RW	-

In the MPAM block:

In the MPAMF_BASE_ns block:

Offset	Name	Description	Access
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO

Offset	Name	Description	Access
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW

Offset	Name	Description	Access
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ

Offset	Name	Description	Access
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW

Offset	Name	Description	Access
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW

Offset	Name	Description	Access
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW

Offset	Name	Description	Access
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW

Offset	Name	Description	Access
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW

In the MPAMF_BASE_rl block:

Offset	Name	Description	Access
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO

Offset	Name	Description	Access
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW

Offset	Name	Description	Access
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ

Offset	Name	Description	Access
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW

Offset	Name	Description	Access
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO

Offset	Name	Description	Access
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO

Offset	Name	Description	Access
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW

In the MPAMF_BASE_rt block:

Offset	Name	Description	Access
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO

Offset	Name	Description	Access
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO

Offset	Name	Description	Access
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW

Offset	Name	Description	Access
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW

Offset	Name	Description	Access
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW

Offset	Name	Description	Access
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW

Offset	Name	Description	Access
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW

Offset	Name	Description	Access
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW

In the MPAMF_BASE_s block:

Offset	Name	Description	Access
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0000	MPAMF_IDR	MPAM Features Identification Register	RO
0x0008	MPAMF_SIDR	MPAM Features Secure Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO

Offset	Name	Description	Access
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0018	MPAMF_IIDR	MPAM Implementation Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0020	MPAMF_AIDR	MPAM Architecture Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0028	MPAMF_IMPL_IDR	MPAM Implementation-Specific Partitioning Feature Identification Register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0030	MPAMF_CPOR_IDR	MPAM Features Cache Portion Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0038	MPAMF_CCAP_IDR	MPAM Features Cache Capacity Partitioning ID register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0040	MPAMF_MBW_IDR	MPAM Memory Bandwidth Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0048	MPAMF_PRI_IDR	MPAM Priority Partitioning Identification Register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0050	MPAMF_PARTID_NRW_IDR	MPAM PARTID Narrowing ID register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0080	MPAMF_MSMON_IDR	MPAM Resource Monitoring Identification Register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO
0x0088	MPAMF_CSUMON_IDR	MPAM Features Cache Storage Usage Monitoring ID register	RO

Offset	Name	Description	Access
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x0090	MPAMF_MBWUMON_IDR	MPAM Features Memory Bandwidth Usage Monitoring ID register	RO
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00DC	MPAMF_ERR_MSI_MPAM	MPAM Error MSI Write MPAM Information Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E0	MPAMF_ERR_MSI_ADDR_L	MPAM Error MSI Low-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E4	MPAMF_ERR_MSI_ADDR_H	MPAM Error MSI High-part Address Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00E8	MPAMF_ERR_MSI_DATA	MPAM Error MSI Data Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00EC	MPAMF_ERR_MSI_ATTR	MPAM Error MSI Write Attributes Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F0	MPAMF_ECR	MPAM Error Control Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x00F8	MPAMF_ESR	MPAM Error Status Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0100	MPAMCFG_PART_SEL	MPAM Partition Configuration Selection Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0108	MPAMCFG_CMAX	MPAM Cache Maximum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW

Offset	Name	Description	Access
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0110	MPAMCFG_CMIN	MPAM Cache Minimum Capacity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0118	MPAMCFG_CASSOC	MPAM Cache Maximum Associativity Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0200	MPAMCFG_MBW_MIN	MPAM Memory Bandwidth Minimum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0208	MPAMCFG_MBW_MAX	MPAM Memory Bandwidth Maximum Partition Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0220	MPAMCFG_MBW_WINWD	MPAM Memory Bandwidth Partitioning Window Width Configuration Register	RW
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0300	MPAMCFG_EN	MPAM Partition Configuration Enable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0310	MPAMCFG_DIS	MPAM Partition Configuration Disable Register	WO/ RAZ
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0320	MPAMCFG_EN_FLAGS	MPAM Partition Configuration Enable Flags Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW

Offset	Name	Description	Access
0x0400	MPAMCFG_PRI	MPAM Priority Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0500	MPAMCFG_MBW_PROP	MPAM Memory Bandwidth Proportional Stride Partition Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0600	MPAMCFG_INTPARTID	MPAM Internal PARTID Narrowing Configuration Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0800	MSMON_CFG_MON_SEL	MPAM Monitor Instance Selection Register	RW
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0808	MSMON_CAPT_EVNT	MPAM Capture Event Generation Register	WO/ RAZ
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0810	MSMON_CFG_CSU_FLT	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0818	MSMON_CFG_CSU_CTL	MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0820	MSMON_CFG_MBWU_FLT	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW

Offset	Name	Description	Access
0x0828	MSMON_CFG_MBWU_CTL	MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0840	MSMON_CSU	MPAM Cache Storage Usage Monitor Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0848	MSMON_CSU_CAPTURE	MPAM Cache Storage Usage Monitor Capture Register	RW
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0858	MSMON_CSU_OFSR	MPAM CSU Monitor Overflow Status Register	RO
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0860	MSMON_MBWU	MPAM Memory Bandwidth Usage Monitor Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0868	MSMON_MBWU_CAPTURE	MPAM Memory Bandwidth Usage Monitor Capture Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0880	MSMON_MBWU_L	MPAM Long Memory Bandwidth Usage Monitor Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0890	MSMON_MBWU_L_CAPTURE	MPAM Long Memory Bandwidth Usage Monitor Capture Register	RW
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x0898	MSMON_MBWU_OFSR	MPAM MBWU Monitor Overflow Status Register	RO
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08DC	MSMON_OFLOW_MSI_MPAM	MPAM Monitor Overflow MSI Write MPAM Information Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW

Offset	Name	Description	Access
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E0	MSMON_OFLOW_MSI_ADDR_L	MPAM Monitor Overflow MSI Low-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E4	MSMON_OFLOW_MSI_ADDR_H	MPAM Monitor Overflow MSI Write High-part Address Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08E8	MSMON_OFLOW_MSI_DATA	MPAM Monitor Overflow MSI Write Data Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08EC	MSMON_OFLOW_MSI_ATTR	MPAM Monitor Overflow MSI Write Attributes Register	RW
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x08F0	MSMON_OFLOW_SR	MPAM Monitor Overflow Status Register	RO
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x1000 + (4 * n)	MPAMCFG_CPBM<n>	MPAM Cache Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>	MPAM Bandwidth Portion Bitmap Partition Configuration Register	RW
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3000	MPAMF_IN_TL_IDR	MPAM Ingress PARTID Translation ID Register	RO
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW
0x3008	MPAMCFG_IN_TL	MPAM Ingress PARTID Translation Configuration Register	RW

Offset	Name	Description	Access
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3010	MPAMCFG_IN_TL_BASE	MPAM Ingress PARTID Translation Base Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3018	MPAMCFG_IN_TL_MASK	MPAM Ingress PARTID Translation Mask Configuration Register	RW
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3200	MPAMF_OUT_TL_IDR	MPAM Egress PARTID Translation ID Register	RO
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3208	MPAMCFG_OUT_TL	MPAM Egress PARTID Translation Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3210	MPAMCFG_OUT_TL_BASE	MPAM Egress PARTID Translation Base Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW
0x3218	MPAMCFG_OUT_TL_MASK	MPAM Egress PARTID Translation Mask Configuration Register	RW

In the PMU block:

Offset	Name	Description	Access	Accessor Co
0x000 + (8 * n) for n in 30:0	PMEVCNTR<n>_EL0	Performance Monitors Event Count Registers	RW	When FEAT_PMUv3_ implemented
0x000 + (8 * n) for n in 30:0	PMEVCNTR<n>_EL0	Performance Monitors Event Count Registers	RW	When FEAT_PMUv3_ implemented and FEAT_PMUv3p implemented
0x000 + (8 * n) for n in 30:0	PMEVCNTR<n>_EL0	Performance Monitors Event Count Registers	RW	When FEAT_PMUv3_ implemented and

Offset	Name	Description	Access	Accessor Co
				FEAT_PMUv3p implemented
0x0F8	PMCCNTR_EL0	Performance Monitors Cycle Counter	RW	When FEAT_PMUv3_ implemented
0x0F8	PMCCNTR_EL0	Performance Monitors Cycle Counter	RW	When FEAT_PMUv3_ implemented
0x0FC	PMCCNTR_EL0[63:32]	Performance Monitors Cycle Counter	RW	When FEAT_PMUv3_ implemented
0x100	PMICNTR_EL0	Performance Monitors Instruction Counter Register	RW	When FEAT_PMUv3_ implemented
0x200	PMPCSR	Program Counter Sample Register	RO	When FEAT_PMUv3_ implemented
0x200	PMPCSR	Program Counter Sample Register	RO	When FEAT_PMUv3_ implemented and FEAT_PCSRv8p implemented
0x204	PMPCSR[63:32]	Program Counter Sample Register	RO	When FEAT_PMUv3_ implemented and FEAT_PCSRv8p implemented
0x208	PMVCIDSR	CONTEXTIDR_EL1 and VMID Sample Register	RO	When FEAT_PMUv3_ implemented
0x208	PMCID1SR	CONTEXTIDR_EL1 Sample Register	RO	When FEAT_PMUv3_ implemented and FEAT_PCSRv8p implemented
0x20C	PMVIDSR	VMID Sample Register	RO	When FEAT_PMUv3_ implemented and FEAT_PCSRv8p implemented
0x220	PMPCSR	Program Counter Sample Register	RO	When FEAT_PMUv3_ implemented
0x220	PMPCSR	Program Counter Sample Register	RO	When FEAT_PMUv3_ implemented and FEAT_PCSRv8p implemented
0x224	PMPCSR[63:32]	Program Counter Sample Register	RO	When FEAT_PMUv3_ implemented and FEAT_PCSRv8p implemented
0x228	PMCCIDSR	CONTEXTIDR_ELx Sample Register	RO	When FEAT_PMUv3_ implemented

Offset	Name	Description	Access	Accessor Co
0x228	PMCID1SR	CONTEXTIDR_EL1 Sample Register	RO	When FEAT_PMUv3_ implemented and FEAT_PCSRv8 implemented
0x22C	PMCID2SR	CONTEXTIDR_EL2 Sample Register	RO	When FEAT_PMUv3_ implemented and FEAT_PCSRv8 implemented
0x400 + (8 * n) for n in 30:0	PMEVTYPER<n>_EL0[63:0]	Performance Monitors Event Type Registers	RW	When FEAT_PMUv3_ implemented
0x400 + (4 * n) for n in 30:0	PMEVTYPER<n>_EL0[31:0]	Performance Monitors Event Type Registers	RW	When FEAT_PMUv3_ implemented
0x47C	PMCCFILTR_EL0[31:0]	Performance Monitors Cycle Counter Filter Register	RW	When FEAT_PMUv3_ implemented
0x480	PMICFILTR_EL0[31:0]	Performance Monitors Instruction Counter Filter Register	RW	When FEAT_PMUv3_ implemented and FEAT_PMUv3_ implemented
0x4F8	PMCCFILTR_EL0	Performance Monitors Cycle Counter Filter Register	RW	When FEAT_PMUv3_ implemented
0x500	PMICFILTR_EL0	Performance Monitors Instruction Counter Filter Register	RW	When FEAT_PMUv3_ implemented and FEAT_PMUv3_ implemented
0x600 + (8 * n) for n in 30:0	PMEVCNTSVR<n>_EL1	Performance Monitors Event Count Saved Value Registers	RO	When FEAT_PM is implemented
0x6F8	PMCCNTSVR_EL1	Performance Monitors Cycle Count Saved Value Register	RO	When FEAT_PM is implemented
0x700	PMICNTSVR_EL1	Performance Monitors Instruction Count Saved Value Register	RO	When FEAT_PM is implemented a FEAT_PMUv3_ implemented
0x800 + (4 * n) for n in 63:0	PMEVFILT2R<n>[31:0]	Performance Monitors Event Filter Registers	RW	When FEAT_PMUv3_ implemented
0x800 + (8 * n) for n in 63:0	PMEVFILT2R<n>[63:0]	Performance Monitors Event Filter Registers	RW	When FEAT_PMUv3_ implemented
0xA00 + (4 * n) for n in 30:0	PMEVTYPER<n>_EL0[63:32]	Performance Monitors Event Type Registers	RW	When FEAT_PMUv3_ implemented and (FEAT_PMUv3 implemented, or FEAT_PMUv3p

Offset	Name	Description	Access	Accessor Co
				implemented, or FEAT_PMUv3_ implemented)
0xA7C	PMCCFILTR_EL0[63:32]	Performance Monitors Cycle Counter Filter Register	RW	When FEAT_PMUv3_ implemented and (FEAT_PMUv3_ implemented, or FEAT_PMUv3p implemented, or FEAT_PMUv3_ implemented)
0xA80	PMICFILTR_EL0[63:32]	Performance Monitors Instruction Counter Filter Register	RW	When FEAT_PMUv3_ implemented and FEAT_PMUv3_ implemented
0xC00	PMCNTENSET_EL0	Performance Monitors Count Enable Set Register	RW	When FEAT_PMUv3_ implemented, or FEAT_PMUv3_ implemented, or FEAT_PMUv3p implemented
0xC00	PMCNTENSET_EL0	Performance Monitors Count Enable Set Register	RW	When FEAT_PMUv3_ implemented, FEAT_PMUv3_ not implemented FEAT_PMUv3p implemented
0xC10	PMCNTEN	Performance Monitors Count Enable register	RW	When FEAT_PMUv3_ implemented
0xC20	PMCNTENCLR_EL0	Performance Monitors Count Enable Clear Register	RW	When FEAT_PMUv3_ implemented, or FEAT_PMUv3_ implemented, or FEAT_PMUv3p implemented
0xC20	PMCNTENCLR_EL0	Performance Monitors Count Enable Clear Register	RW	When FEAT_PMUv3_ implemented, FEAT_PMUv3_ not implemented FEAT_PMUv3p implemented
0xC40	PMINTENSET_EL1	Performance Monitors Interrupt Enable Set Register	RW	When FEAT_PMUv3_ implemented, or FEAT_PMUv3_ implemented, or FEAT_PMUv3p implemented
0xC40	PMINTENSET_EL1	Performance Monitors Interrupt Enable Set Register	RW	When FEAT_PMUv3_ implemented, FEAT_PMUv3_ not implemented FEAT_PMUv3p implemented
0xC50	PMINTEN	Performance Monitors Interrupt Enable register	RW	When FEAT_PMUv3_ implemented

Offset	Name	Description	Access	Accessor Co
0xC60	PMINTENCLR_EL1	Performance Monitors Interrupt Enable Clear Register	RW	When FEAT_PMUv3_ implemented, or FEAT_PMUv3_ implemented, or FEAT_PMUv3p implemented
0xC60	PMINTENCLR_EL1	Performance Monitors Interrupt Enable Clear Register	RW	When FEAT_PMUv3_ implemented, FEAT_PMUv3_ not implemented FEAT_PMUv3p implemented
0xC80	PMOVSCLR_EL0	Performance Monitors Overflow Flag Status Clear register	RW	When FEAT_PMUv3_ implemented, or FEAT_PMUv3_ implemented, or FEAT_PMUv3p implemented
0xC80	PMOVSCLR_EL0	Performance Monitors Overflow Flag Status Clear register	RW	When FEAT_PMUv3_ implemented, FEAT_PMUv3_ not implemented FEAT_PMUv3p implemented
0xC90	PMOVS	Performance Monitors Overflow Flag Status register	RW	When FEAT_PMUv3_ implemented
0xCA0	PMSWINC_EL0	Performance Monitors Software Increment Register	WO	When FEAT_PMUv3_ implemented and FEAT_PMUv3p implemented
0xCA0	PMZR_EL0	Performance Monitors Zero with Mask	WO	When FEAT_PMUv3_ implemented and FEAT_PMUv3p implemented
0xCC0	PMOVSSET_EL0	Performance Monitors Overflow Flag Status Set Register	RW	When FEAT_PMUv3_ implemented, or FEAT_PMUv3_ implemented, or FEAT_PMUv3p implemented
0xCC0	PMOVSSET_EL0	Performance Monitors Overflow Flag Status Set Register	RW	When FEAT_PMUv3_ implemented, FEAT_PMUv3_ not implemented FEAT_PMUv3p implemented
0xCE0	PMCGCR0	Counter Group Configuration Register 0	RO	When FEAT_PMUv3_ implemented and FEAT_PMUv3_ implemented
0xCE0	PMCGCR0	Counter Group Configuration Register 0	RO	When FEAT_PMUv3_ implemented and

Offset	Name	Description	Access	Accessor Co
				FEAT_PMUv3_ implemented
0xE00	PMCFGR	Performance Monitors Configuration Register	RO	When FEAT_PMUv3_ implemented
0xE00	PMCFGR	Performance Monitors Configuration Register	RO	When FEAT_PMUv3_ implemented
0xE04	PMCR_EL0	Performance Monitors Control Register	RW	When FEAT_PMUv3_ implemented
0xE08	PMIIDR	Performance Monitors Implementation Identification Register	RO	When FEAT_PMUv3_ implemented
0xE10	PMCR_EL0	Performance Monitors Control Register	RW	When FEAT_PMUv3_ implemented
0xE20	PMCEID0	Performance Monitors Common Event Identification register 0	RO	When FEAT_PMUv3_ implemented
0xE24	PMCEID1	Performance Monitors Common Event Identification register 1	RO	When FEAT_PMUv3_ implemented
0xE28	PMCEID2	Performance Monitors Common Event Identification register 2	RO	When FEAT_PMUv3_ implemented and FEAT_PMUv3p implemented
0xE2C	PMCEID3	Performance Monitors Common Event Identification register 3	RO	When FEAT_PMUv3_ implemented and FEAT_PMUv3p implemented
0xE30	PMSSCR_EL1	Performance Monitors Snapshot Status and Capture Register	RW	When FEAT_PM is implemented
0xE40	PMMIR	Performance Monitors Machine Identification Register	RO	When FEAT_PM implemented and (FEAT_PMUv3_ is implemented FEAT_PMUv3p implemented)
0xE40	PMMIR	Performance Monitors Machine Identification Register	RO	When FEAT_PM implemented, FEAT_PMUv3_ implemented, an FEAT_PMUv3p implemented
0xE50	PMPCSCTL	PC Sample-based Profiling Control Register	RW	When FEAT_PC is implemented
0xE58	PMCCR	PMU Configuration Control Register	RW	When FEAT_PMUv3_ is implemented

Offset	Name	Description	Access	Accessor Co
0xF00	PMITCTRL	Performance Monitors Integration mode Control register	RW	When FEAT_PMUv3_ implemented
0xFA8	PMDEVAFF	Performance Monitors Device Affinity register	RO	When FEAT_PMUv3_ implemented
0xFA8	PMDEVAFF0	Performance Monitors Device Affinity register 0	RO	When FEAT_PMUv3_ implemented
0xFAC	PMDEVAFF1	Performance Monitors Device Affinity register 1	RO	When FEAT_PMUv3_ implemented
0xFB0	PMLAR	Performance Monitors Lock Access Register	WO	When FEAT_PMUv3_ implemented
0xFB4	PMLSR	Performance Monitors Lock Status Register	RO	When FEAT_PMUv3_ implemented
0xFB8	PMAUTHSTATUS	Performance Monitors Authentication Status register	RO	When FEAT_PMUv3_ implemented
0xFBC	PMDEVARCH	Performance Monitors Device Architecture register	RO	When FEAT_PMUv3_ implemented
0xFC8	PMDEVID	Performance Monitors Device ID register	RO	When FEAT_PMUv3_ implemented an or FEAT_PCSR implemented)
0xFCC	PMDEVTYPE	Performance Monitors Device Type register	RO	When FEAT_PMUv3_ implemented
0xFD0	PMPIDR4	Performance Monitors Peripheral Identification Register 4	RO	When FEAT_PMUv3_ implemented
0xFE0	PMPIDR0	Performance Monitors Peripheral Identification Register 0	RO	When FEAT_PMUv3_ implemented
0xFE4	PMPIDR1	Performance Monitors Peripheral Identification Register 1	RO	When FEAT_PMUv3_ implemented
0xFE8	PMPIDR2	Performance Monitors Peripheral Identification Register 2	RO	When FEAT_PMUv3_ implemented
0xFEC	PMPIDR3	Performance Monitors Peripheral Identification Register 3	RO	When FEAT_PMUv3_ implemented

Offset	Name	Description	Access	Accessor Co
0xFF0	PMCIDR0	Performance Monitors Component Identification Register 0	RO	When FEAT_PMUv3_ implemented
0xFF4	PMCIDR1	Performance Monitors Component Identification Register 1	RO	When FEAT_PMUv3_ implemented
0xFF8	PMCIDR2	Performance Monitors Component Identification Register 2	RO	When FEAT_PMUv3_ implemented
0xFFC	PMCIDR3	Performance Monitors Component Identification Register 3	RO	When FEAT_PMUv3_ implemented

In the RAS block:

Offset	Name	Description	Access	
0x000 + (64 * n)	ERR<n>FR	Error Record <n> Feature Register	RO	-
0x008 + (64 * n)	ERR<n>CTLR	Error Record <n> Control Register	RW	-
0x010 + (64 * n)	ERR<n>STATUS	Error Record <n> Primary Status Register	RW	-
0x018 + (64 * n)	ERR<n>ADDR	Error Record <n> Address Register	RW	-
0x020 + (64 * n)	ERR<n>MISC0	Error Record <n> Miscellaneous Register 0	RW	-
0x028 + (64 * n)	ERR<n>MISC1	Error Record <n> Miscellaneous Register 1	RW	-
0x030 + (64 * n)	ERR<n>MISC2	Error Record <n> Miscellaneous Register 2	RW	-
0x038 + (64 * n)	ERR<n>MISC3	Error Record <n> Miscellaneous Register 3	RW	-
0x800 + (64 * n)	ERR<n>PFGF	Error Record <n> Pseudo-fault Generation Feature Register	RO	-
0x800 + (8 * n)	ERRIMPDEF<n>	IMPLEMENTATION DEFINED Register <n>	RW	-
0x808 + (64 * n)	ERR<n>PFGCTL	Error Record <n> Pseudo-fault Generation Control Register	RW	-
0x810 + (64 * n)	ERR<n>PFGCDN	Error Record <n> Pseudo-fault Generation Countdown Register	RW	-
0xE00 + (64 * m)	ERRGSR<m>	Error Group <n> Status Register	RO	-
0xE10	ERRIIDR	Implementation Identification Register	RO	-
0xE40	ERRACR	Access Configuration Register	RW	-
0xE80	ERRFHICR0	Fault Handling Interrupt Configuration Register 0	RW	-
0xE80 + (8 * n)	ERRIRQCR<n>	Generic Error Interrupt Configuration Register <n>	RW	-
0xE88	ERRFHICR1	Fault Handling Interrupt Configuration Register 1	RW	-
0xE8C	ERRFHICR2	Fault Handling Interrupt Configuration Register 2	RW	-
0xE90	ERRERICR0	Error Recovery Interrupt Configuration Register 0	RW	-
0xE98	ERRERICR1	Error Recovery Interrupt Configuration Register 1	RW	-
0xE9C	ERRERICR2	Error Recovery Interrupt Configuration Register 2	RW	-
0xEA0	ERRCRICR0	Critical Error Interrupt Configuration Register 0	RW	-
0xEA8	ERRCRICR1	Critical Error Interrupt Configuration Register 1	RW	-
0xEAC	ERRCRICR2	Critical Error Interrupt Configuration Register 2	RW	-
0xEF8	ERRIRQSR	Error Interrupt Status Register	RW	-
0xFA8	ERRDEVAFF	Device Affinity Register	RO	-
0xFBC	ERRDEVARCH	Device Architecture Register	RO	-
0xFC8	ERRDEVID	Device Configuration Register	RO	-
0xFD0	ERRPIDR4	Peripheral Identification Register 4	RO	-

Offset	Name	Description	Access	
0xFE0	ERRPIDR0	Peripheral Identification Register 0	RO	-
0xFE4	ERRPIDR1	Peripheral Identification Register 1	RO	-
0xFE8	ERRPIDR2	Peripheral Identification Register 2	RO	-
0xFEC	ERRPIDR3	Peripheral Identification Register 3	RO	-
0xFF0	ERRCIDR0	Component Identification Register 0	RO	-
0xFF4	ERRCIDR1	Component Identification Register 1	RO	-
0xFF8	ERRCIDR2	Component Identification Register 2	RO	-
0xFFC	ERRCIDR3	Component Identification Register 3	RO	-

In the TRBE block:

Offset	Name	Description	Access	
0x000	TRBBASER_EL1	Trace Buffer Base Address Register	RW	-
0x008	TRBPTR_EL1	Trace Buffer Write Pointer Register	RW	-
0x010	TRBLIMITR_EL1	Trace Buffer Limit Address Register	RW	-
0x018	TRBSR_EL1	Trace Buffer Status/syndrome Register	RW	-
0x020	TRBTRG_EL1	Trace Buffer Trigger Counter Register	RW	-
0x028	TRBMAR_EL1	Trace Buffer Memory Attribute Register	RW	-
0x030	TRBIDR_EL1	Trace Buffer ID Register	RO	-
0x038	TRBCR	Trace Buffer Control Register	RW	-
0x040	TRBMPAM_EL1	Trace Buffer MPAM Configuration Register	RW	-
0xF00	TRBITCTRL	Integration Mode Control Register	RW	-
0xFA8	TRBDEVAFF	Device Affinity Register	RO	-
0xFB0	TRBLAR	Lock Access Register	WO	-
0xFB4	TRBLSR	Lock Status Register	RO	-
0xFB8	TRBAUTHSTATUS	Authentication Status Register	RO	-
0xFBC	TRBDEVARCH	Trace Buffer Device Architecture Register	RO	-
0xFC0	TRBDEVID2	Device Configuration Register 2	RO	-
0xFC4	TRBDEVID1	Device Configuration Register 1	RO	-
0xFC8	TRBDEVID	Device Configuration Register	RO	-
0xFCC	TRBDEVTYPE	Device Type Register	RO	-
0xFD0	TRBPIDR4	Peripheral Identification Register 4	RO	-
0xFD4	TRBPIDR5	Peripheral Identification Register 5	RO	-
0xFD8	TRBPIDR6	Peripheral Identification Register 6	RO	-
0xFDC	TRBPIDR7	Peripheral Identification Register 7	RO	-
0xFE0	TRBPIDR0	Peripheral Identification Register 0	RO	-
0xFE4	TRBPIDR1	Peripheral Identification Register 1	RO	-
0xFE8	TRBPIDR2	Peripheral Identification Register 2	RO	-
0xFEC	TRBPIDR3	Peripheral Identification Register 3	RO	-
0xFF0	TRBCIDR0	Component Identification Register 0	RO	-
0xFF4	TRBCIDR1	Component Identification Register 1	RO	-
0xFF8	TRBCIDR2	Component Identification Register 2	RO	-
0xFFC	TRBCIDR3	Component Identification Register 3	RO	-

In the Timer block:

In the CNTBaseN block:

Offset	Name	Description	Access
0x000	CNTPCT[31:0]	Counter-timer Physical Count	RO
0x000	CNTPCT[31:0]	Counter-timer Physical Count	RO
0x000	CNTPCT[31:0]	Counter-timer Physical Count	RO
0x000	CNTPCT[31:0]	Counter-timer Physical Count	RO
0x004	CNTPCT[63:32]	Counter-timer Physical Count	RO
0x004	CNTPCT[63:32]	Counter-timer Physical Count	RO
0x004	CNTPCT[63:32]	Counter-timer Physical Count	RO
0x004	CNTPCT[63:32]	Counter-timer Physical Count	RO
0x008	CNTVCT[31:0]	Counter-timer Virtual Count	RO
0x008	CNTVCT[31:0]	Counter-timer Virtual Count	RO
0x008	CNTVCT[31:0]	Counter-timer Virtual Count	RO
0x008	CNTVCT[31:0]	Counter-timer Virtual Count	RO
0x00C	CNTVCT[63:32]	Counter-timer Virtual Count	RO
0x00C	CNTVCT[63:32]	Counter-timer Virtual Count	RO
0x00C	CNTVCT[63:32]	Counter-timer Virtual Count	RO
0x00C	CNTVCT[63:32]	Counter-timer Virtual Count	RO
0x010	CNTFRQ	Counter-timer Frequency	RO
0x010	CNTFRQ	Counter-timer Frequency	RO
0x010	CNTFRQ	Counter-timer Frequency	RO
0x014	CNTEL0ACR	Counter-timer EL0 Access Control Register	RW
0x018	CNTVOFF[31:0]	Counter-timer Virtual Offset	RO
0x018	CNTVOFF[31:0]	Counter-timer Virtual Offset	RO
0x01C	CNTVOFF[63:32]	Counter-timer Virtual Offset	RO
0x01C	CNTVOFF[63:32]	Counter-timer Virtual Offset	RO
0x020	CNTP_CVAL[31:0]	Counter-timer Physical Timer CompareValue	RW
0x020	CNTP_CVAL[31:0]	Counter-timer Physical Timer CompareValue	RW
0x020	CNTP_CVAL[31:0]	Counter-timer Physical Timer CompareValue	RW
0x020	CNTP_CVAL[31:0]	Counter-timer Physical Timer CompareValue	RW
0x024	CNTP_CVAL[63:32]	Counter-timer Physical Timer CompareValue	RW
0x024	CNTP_CVAL[63:32]	Counter-timer Physical Timer CompareValue	RW
0x024	CNTP_CVAL[63:32]	Counter-timer Physical Timer CompareValue	RW
0x024	CNTP_CVAL[63:32]	Counter-timer Physical Timer CompareValue	RW
0x028	CNTP_TVAL	Counter-timer Physical Timer TimerValue	RW
0x028	CNTP_TVAL	Counter-timer Physical Timer TimerValue	RW
0x02C	CNTP_CTL	Counter-timer Physical Timer Control	RW
0x02C	CNTP_CTL	Counter-timer Physical Timer Control	RW
0x030	CNTV_CVAL[31:0]	Counter-timer Virtual Timer CompareValue	RW
0x030	CNTV_CVAL[31:0]	Counter-timer Virtual Timer CompareValue	RW
0x030	CNTV_CVAL[31:0]	Counter-timer Virtual Timer CompareValue	RW
0x030	CNTV_CVAL[31:0]	Counter-timer Virtual Timer CompareValue	RW
0x034	CNTV_CVAL[63:32]	Counter-timer Virtual Timer CompareValue	RW
0x034	CNTV_CVAL[63:32]	Counter-timer Virtual Timer CompareValue	RW
0x034	CNTV_CVAL[63:32]	Counter-timer Virtual Timer CompareValue	RW
0x034	CNTV_CVAL[63:32]	Counter-timer Virtual Timer CompareValue	RW
0x038	CNTV_TVAL	Counter-timer Virtual Timer TimerValue	RW

Offset	Name	Description	Access
0x038	CNTV_TVAL	Counter-timer Virtual Timer TimerValue	RW
0x03C	CNTV_CTL	Counter-timer Virtual Timer Control	RW
0x03C	CNTV_CTL	Counter-timer Virtual Timer Control	RW
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO

In the CNTCTLBase block:

Offset	Name	Description	Access
0x000	CNTFRQ	Counter-timer Frequency	RO
0x000	CNTFRQ	Counter-timer Frequency	RO
0x000	CNTFRQ	Counter-timer Frequency	RO
0x004	CNTNSAR	Counter-timer Non-secure Access Register	RW
0x008	CNTTIDR	Counter-timer Timer ID Register	RO
0x040 + (4 * n)	CNTACR<n>	Counter-timer Access Control Registers	RW
0x080 + (8 * n)	CNTVOFF<n>[31:0]	Counter-timer Virtual Offsets	RW
0x080 + (8 * n)	CNTVOFF<n>[31:0]	Counter-timer Virtual Offsets	RW
0x084 + (8 * n)	CNTVOFF<n>[63:32]	Counter-timer Virtual Offsets	RW
0x084 + (8 * n)	CNTVOFF<n>[63:32]	Counter-timer Virtual Offsets	RW
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO

In the CNTControlBase block:

Offset	Name	Description	Access
0x000	CNTCR	Counter Control Register	RW
0x004	CNTSR	Counter Status Register	RO
0x008	CNTCV[63:0]	Counter Count Value register	RW
0x008	CNTCV[63:0]	Counter Count Value register	RW
0x020	CNTFID0	Counter Frequency ID	ImplementationDefined:RO,RW
0x020 + (4 * n)	CNTFID<n>	Counter Frequency IDs, n > 0	ImplementationDefined:RO,RW
0x10	CNTSCR	Counter Scale Register	RW
0x1C	CNTID	Counter Identification Register	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO

In the CNTEL0BaseN block:

Offset	Name	Description	Access
0x000	CNTPCT[31:0]	Counter-timer Physical Count	RO

Offset	Name	Description	Access
0x000	CNTPCT[31:0]	Counter-timer Physical Count	RO
0x000	CNTPCT[31:0]	Counter-timer Physical Count	RO
0x000	CNTPCT[31:0]	Counter-timer Physical Count	RO
0x004	CNTPCT[63:32]	Counter-timer Physical Count	RO
0x004	CNTPCT[63:32]	Counter-timer Physical Count	RO
0x004	CNTPCT[63:32]	Counter-timer Physical Count	RO
0x004	CNTPCT[63:32]	Counter-timer Physical Count	RO
0x008	CNTVCT[31:0]	Counter-timer Virtual Count	RO
0x008	CNTVCT[31:0]	Counter-timer Virtual Count	RO
0x008	CNTVCT[31:0]	Counter-timer Virtual Count	RO
0x008	CNTVCT[31:0]	Counter-timer Virtual Count	RO
0x00C	CNTVCT[63:32]	Counter-timer Virtual Count	RO
0x00C	CNTVCT[63:32]	Counter-timer Virtual Count	RO
0x00C	CNTVCT[63:32]	Counter-timer Virtual Count	RO
0x00C	CNTVCT[63:32]	Counter-timer Virtual Count	RO
0x010	CNTFRQ	Counter-timer Frequency	RO
0x010	CNTFRQ	Counter-timer Frequency	RO
0x010	CNTFRQ	Counter-timer Frequency	RO
0x020	CNTP_CVAL[31:0]	Counter-timer Physical Timer CompareValue	RW
0x020	CNTP_CVAL[31:0]	Counter-timer Physical Timer CompareValue	RW
0x020	CNTP_CVAL[31:0]	Counter-timer Physical Timer CompareValue	RW
0x020	CNTP_CVAL[31:0]	Counter-timer Physical Timer CompareValue	RW
0x024	CNTP_CVAL[63:32]	Counter-timer Physical Timer CompareValue	RW
0x024	CNTP_CVAL[63:32]	Counter-timer Physical Timer CompareValue	RW
0x024	CNTP_CVAL[63:32]	Counter-timer Physical Timer CompareValue	RW
0x024	CNTP_CVAL[63:32]	Counter-timer Physical Timer CompareValue	RW
0x028	CNTP_TVAL	Counter-timer Physical Timer TimerValue	RW
0x028	CNTP_TVAL	Counter-timer Physical Timer TimerValue	RW
0x02C	CNTP_CTL	Counter-timer Physical Timer Control	RW
0x02C	CNTP_CTL	Counter-timer Physical Timer Control	RW
0x030	CNTV_CVAL[31:0]	Counter-timer Virtual Timer CompareValue	RW
0x030	CNTV_CVAL[31:0]	Counter-timer Virtual Timer CompareValue	RW
0x030	CNTV_CVAL[31:0]	Counter-timer Virtual Timer CompareValue	RW
0x030	CNTV_CVAL[31:0]	Counter-timer Virtual Timer CompareValue	RW
0x034	CNTV_CVAL[63:32]	Counter-timer Virtual Timer CompareValue	RW
0x034	CNTV_CVAL[63:32]	Counter-timer Virtual Timer CompareValue	RW
0x034	CNTV_CVAL[63:32]	Counter-timer Virtual Timer CompareValue	RW
0x034	CNTV_CVAL[63:32]	Counter-timer Virtual Timer CompareValue	RW
0x038	CNTV_TVAL	Counter-timer Virtual Timer TimerValue	RW
0x038	CNTV_TVAL	Counter-timer Virtual Timer TimerValue	RW
0x03C	CNTV_CTL	Counter-timer Virtual Timer Control	RW
0x03C	CNTV_CTL	Counter-timer Virtual Timer Control	RW
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO

In the CNTReadBase block:

Offset	Name	Description	Access
0x000	CNTCV[63:0]	Counter Count Value register	RO
0x000	CNTCV[63:0]	Counter Count Value register	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO
0xFD0 + (4 * n)	CounterID<n>	Counter ID registers	RO

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ASICCTL, CTI External Multiplexer Control register

The ASICCTL characteristics are:

Purpose

Can be used to provide IMPLEMENTATION DEFINED controls for the CTI. For example, the register might be used to control multiplexors for additional IMPLEMENTATION DEFINED triggers. The IMPLEMENTATION DEFINED controls provided by this register might modify the architecturally defined behavior of the CTI.

Note

The architecturally-defined triggers must not be multiplexed.

Configuration

It is IMPLEMENTATION DEFINED whether ASICCTL is implemented in the Core power domain or in the Debug power domain.

If it is implemented in the Core power domain then it is IMPLEMENTATION DEFINED whether it is in the Cold reset domain or the Warm reset domain.

This register must reset to a value that supports the architecturally-defined behavior of the CTI. Changing the value of the register from its reset value causes IMPLEMENTATION DEFINED behavior that might differ from the architecturally-defined behavior of the CTI.

Other than the requirements listed in this register description, all aspects of the reset behavior of the ASICCTL are IMPLEMENTATION DEFINED.

Attributes

ASICCTL is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMPLEMENTATION DEFINED																															

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ASICCTL

ASICCTL can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x144	ASICCTL

Accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess(addrdesc), and SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **IMPDEF**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTACR<n>, Counter-timer Access Control Registers, n = 0 - 7

The CNTACR<n> characteristics are:

Purpose

Provides top-level access controls for the elements of a timer frame. CNTACR<n> provides the controls for frame CNTBaseN.

In addition to the CNTACR<n> control:

- CNTNSAR controls whether CNTACR<n> is accessible by Non-secure accesses.
- If frame CNTEL0BaseN is implemented, the CNTEL0ACR in frame CNTBaseN provides additional control of accesses to frame CNTEL0BaseN.

Configuration

It is IMPLEMENTATION DEFINED whether CNTACR<n> is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Implemented only if the value of CNTTIDR.Frame<n> is 1.

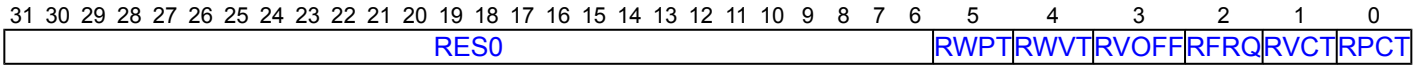
An implementation of the counters might not provide configurable access to some or all of the features. In this case, the associated field in the CNTACR<n> register is:

- RAZ/WI if access is always denied.
- RAO/WI if access is always permitted.

Attributes

CNTACR<n> is a 32-bit register.

Field descriptions



Bits [31:6]

Reserved, RES0.

RWPT, bit [5]

Read/write access to the EL1 Physical Timer registers CNTP_CVAL, CNTP_TVAL, and CNTP_CTL, in frame <n>.

RWPT	Meaning
0b0	No access to the EL1 Physical Timer registers in frame <n>. The registers are RES0.
0b1	Read/write access to the EL1 Physical Timer registers in frame <n>.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

RWVT, bit [4]

Read/write access to the Virtual Timer register CNTV_CVAL, CNTV_TVAL, and CNTV_CTL, in frame <n>.

RWVT	Meaning
0b0	No access to the Virtual Timer registers in frame <n>. The registers are RES0.
0b1	Read/write access to the Virtual Timer registers in frame <n>.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

RVOFF, bit [3]

Read-only access to CNTVOFF, in frame <n>.

RVOFF	Meaning
0b0	No access to CNTVOFF in frame <n>. The register is RES0.
0b1	Read-only access to CNTVOFF in frame <n>.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

RFRQ, bit [2]

Read-only access to CNTFRQ, in frame <n>.

RFRQ	Meaning
0b0	No access to CNTFRQ in frame <n>. The register is RES0.
0b1	Read-only access to CNTFRQ in frame <n>.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

RVCT, bit [1]

Read-only access to CNTVCT, in frame <n>.

RVCT	Meaning
0b0	No access to CNTVCT in frame <n>. The register is RES0.
0b1	Read-only access to CNTVCT in frame <n>.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

RPCT, bit [0]

Read-only access to CNTPCT, in frame <n>.

RPCT	Meaning
0b0	No access to CNTPCT in frame <n>. The register is RES0.
0b1	Read-only access to CNTPCT in frame <n>.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTACR<n>

In a system that supports the Realm Management Extension, CNTNSAR.NS<n> describes how these registers can be accessed by Root or Realm accesses.

In a system that recognizes two Security states:

- CNTACR<n> is always accessible by Secure accesses.

- CNTNSAR.NS<n> determines whether CNTACR<n> is accessible by Non-secure accesses.

CNTACR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTCTLBase	`0x040` + (4 * n)	CNTACR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTCR, Counter Control Register

The CNTCR characteristics are:

Purpose

Enables the counter, controls the counter frequency setting, and controls counter behavior during debug.

Configuration

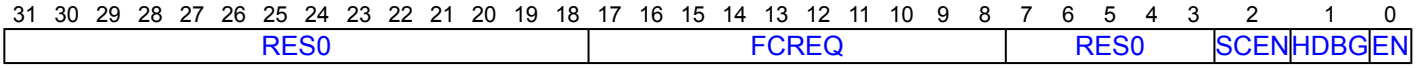
It is IMPLEMENTATION DEFINED whether CNTCR is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTCR is a 32-bit register.

Field descriptions



Bits [31:18]

Reserved, RES0.

FCREQ, bits [17:8]

Frequency change request. Indicates the number of the entry in the Frequency modes table to select.

Selecting an unimplemented entry, or an entry that contains 0, has no effect on the counter.

The maximum number of entries in the Frequency modes table is IMPLEMENTATION DEFINED up to a maximum of 1004 entries, see 'The Frequency modes table'. An implementation is only required to implement an FCREQ field that can hold values from 0 to the highest supported Frequency modes table entry. Any unrequired most-significant bits of FCREQ can be implemented as RES0.

The reset behavior of this field is:

- On a Timer reset, this field resets to `'0000000000'`.

Bits [7:3]

Reserved, RES0.

SCEN, bit [2]

When FEAT_CNTSC is implemented:

Scale Enable.

SCEN	Meaning
`0b0`	Scaling is not enabled. The counter value is incremented by `0x1`.0000000 for each counter tick.
`0b1`	Scaling is enabled. The counter is incremented by CNTSCR.ScaleVal for each counter tick.

The SCEN bit can only be changed when the counter is disabled, when CNTCR.EN == 0.

If the value of CNTCR.SCEN changes when CNTCR.EN == 1 then:

- The counter value becomes UNKNOWN.
- The counter value remains UNKNOWN on future ticks of the clock.

When the CNTCV register in the CNTControlBase frame of the memory mapped counter module is written to, the accumulated fraction information is reset to zero.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HDBG, bit [1]

Halt-on-debug. Controls whether a Halt-on-debug signal halts the system counter:

HDBG	Meaning
0b0	System counter ignores Halt-on-debug.
0b1	Asserted Halt-on-debug signal halts system counter update.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

EN, bit [0]

Enables the counter:

EN	Meaning
0b0	System counter disabled.
0b1	System counter enabled.

The reset behavior of this field is:

- On a Timer reset, this field resets to '0'.

Accessing CNTCR

In a system that supports the Realm Management Extension, the CNTControlBase frame, which includes this register, is implemented only in the Root physical address space.

In a system that supports Secure and Non-secure physical address spaces, the CNTControlBase frame, which includes this register, is implemented only in the Secure physical address space.

CNTCR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTControlBase	0x000	CNTCR

Accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTCV, Counter Count Value register

The CNTCV characteristics are:

Purpose

Indicates the current count value.

Configuration

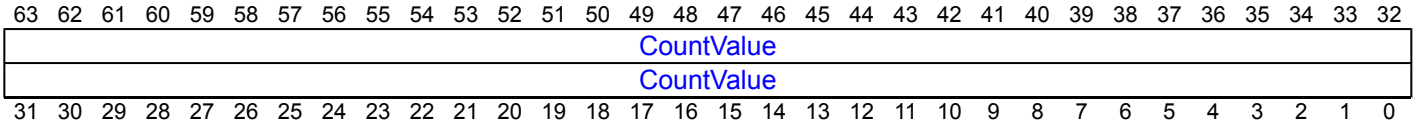
It is IMPLEMENTATION DEFINED whether CNTCV is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTCV is a 64-bit register.

Field descriptions



CountValue, bits [63:0]

Indicates the counter value.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTCV

Frame	Accessibility
CNTControlBase	RW
CNTReadBase	RO

A write to CNTCV must be visible in the CNTPCT register of each running processor in a finite time.

For the instance of the register in the CNTControlBase frame:

- In a system that supports the Realm Management Extension, this register is implemented only in the Root physical address space.
- In a system that supports Secure and Non-secure physical address spaces, this register is implemented only in the Secure physical address space.
- If the counter is enabled, the effect of writing to the register is UNKNOWN.

For the instance of the register in the CNTReadBase frame, this register is accessible in all physical address spaces.

In an implementation that supports 64-bit atomic memory accesses, this register must be accessible using a 64-bit atomic access.

CNTCV can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance	Range
Timer	CNTControlBase	0x008	CNTCV	63:0

Accesses to this register are **RW**.

Component	Frame	Offset	Instance	Range
Timer	CNTReadBase	0x000	CNTCV	63:0

Accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTEL0ACR, Counter-timer EL0 Access Control Register

The CNTEL0ACR characteristics are:

Purpose

An implementation of CNTEL0ACR in the frame at CNTBaseN controls whether the CNTPCT, CNTVCT, CNTFRQ, EL1 Physical Timer, and Virtual Timer registers are visible in the frame at CNTEL0BaseN.

Configuration

It is IMPLEMENTATION DEFINED whether CNTEL0ACR is implemented in the Core power domain or in the Debug power domain.

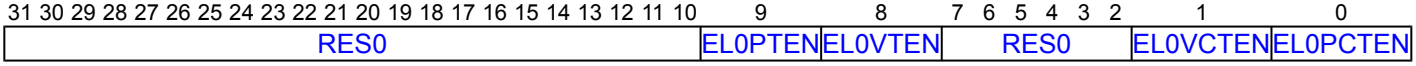
Implementation of this register is OPTIONAL.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTEL0ACR is a 32-bit register.

Field descriptions



Bits [31:10]

Reserved, RES0.

EL0PTEN, bit [9]

Second view read/write access control for the EL1 Physical Timer registers. This bit controls whether the CNTP_CVAL, CNTP_TVAL, and CNTP_CTL registers in the current CNTBaseN frame are also accessible in the corresponding CNTEL0BaseN frame.

EL0PTEN	Meaning
0b0	No access. Registers are RES0 in the second view.
0b1	Access permitted. If the registers are accessible in the current frame then they are accessible in the second view.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

EL0VTEN, bit [8]

Second view read/write access control for the Virtual Timer registers. This bit controls whether the CNTV_CVAL, CNTV_TVAL, and CNTV_CTL registers in the current CNTBaseN frame are also accessible in the corresponding CNTEL0BaseN frame.

EL0VTEN	Meaning
0b0	No access. Registers are RES0 in the second view.
0b1	Access permitted. If the registers are accessible in the current frame then they are accessible in the second view.

The definition of this bit means that, if the Virtual Timer registers are not implemented in the current CNTBaseN frame, then the Virtual Timer register addresses are RES0 in the corresponding CNTEL0BaseN frame, regardless of the value of this bit.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Bits [7:2]

Reserved, RES0.

EL0VCTEN, bit [1]

Second view read access control for CNTVCT and CNTFRQ.

EL0VCTEN	Meaning
0b0	CNTVCT is not visible in the second view. If EL0PCTEN is set to 0, CNTFRQ is not visible in the second view.
0b1	Access permitted. If CNTVCT and CNTFRQ are visible in the current frame then they are visible in the second view.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

EL0PCTEN, bit [0]

Second view read access control for CNTPCT and CNTFRQ.

EL0PCTEN	Meaning
0b0	CNTPCT is not visible in the second view. If EL0VCTEN is set to 0, CNTFRQ is not visible in the second view.
0b1	Access permitted. If CNTPCT and CNTFRQ are visible in the current frame then they are visible in the second view.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTEL0ACR

CNTEL0ACR can be implemented in any implemented CNTBaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

If CNTEL0ACR is not implemented in an implemented CNTBaseN frame:

- The register location in that frame is RAZ/WI.
- If the corresponding CNTEL0BaseN frame is implemented, the registers CNTFRQ, CNTP_CTL, CNTP_CVAL, CNTP_TVAL, CNTPCT, CNTV_CTL, CNTV_CVAL, CNTV_TVAL, and CNTVCT are not visible and accesses are RAZ/WI in that frame.

CNTEL0ACR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTBaseN	0x014	CNTEL0ACR

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTFID0, Counter Frequency ID

The CNTFID0 characteristics are:

Purpose

Indicates the base frequency of the system counter.

Configuration

It is IMPLEMENTATION DEFINED whether CNTFID0 is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

The possible frequencies for the system counter are stored in the Frequency modes table as 32-bit words starting with the base frequency, CNTFID0. For more information, see 'The Frequency modes table'.

The final entry in the Frequency modes table must be followed by a 32-bit word of zero value, to mark the end of the table.

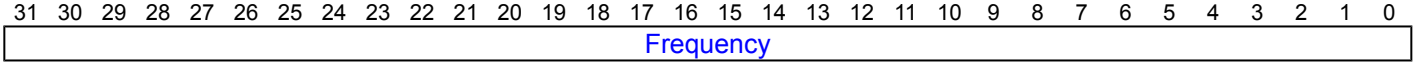
Typically, the Frequency modes table will be in read-only memory. However, a system implementation might use read/write memory for the table, and initialize the table entries as part of its start-up sequence.

If the Frequency modes table is in read/write memory, Arm strongly recommends that the table is not updated once the system is running.

Attributes

CNTFID0 is a 32-bit register.

Field descriptions



Frequency, bits [31:0]

The base frequency of the system counter, in Hz.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFID0

It is IMPLEMENTATION DEFINED whether this register is RO or RW

In a system that supports the Realm Management Extension, the CNTControlBase frame, which includes this register, is implemented only in the Root physical address space.

In a system that supports Secure and Non-secure physical address spaces, the CNTControlBase frame, which includes this register, is implemented only in the Secure physical address space.

CNTFID0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTControlBase	0x020	CNTFID0

Accesses to this register are **RO or RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTFID<n>, Counter Frequency IDs, n > 0, n = 1 - 1003

The CNTFID<n> characteristics are:

Purpose

Indicates alternative system counter update frequencies.

Configuration

It is IMPLEMENTATION DEFINED whether CNTFID<n> is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

The possible frequencies for the system counter are stored in the Frequency modes table as 32-bit words starting with the base frequency, CNTFID0, see 'The Frequency modes table'.

The number of CNTFID<n> registers is IMPLEMENTATION DEFINED, and the only required CNTFID<n> register is CNTFID0.

The final entry in the Frequency modes table must be followed by a 32-bit word of zero value, to mark the end of the table.

The architecture can support up to 1004 entries in the Frequency modes table, including the zero-word end marker, and the number of entries is IMPLEMENTATION DEFINED up to this limit. For an implementation that includes registers in the IMPLEMENTATION DEFINED register space `0x0C0-0x0FC`, the maximum number of entries in the Frequency modes table is 40, including the zero-word end marker.

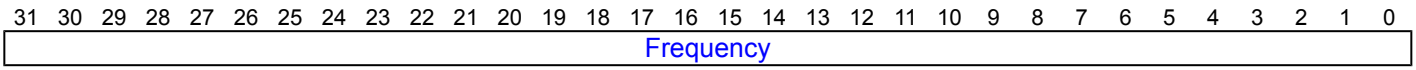
Typically, the Frequency modes table will be in read-only memory. However, a system implementation might use read/write memory for the table, and initialize the table entries as part of its start-up sequence.

If the Frequency modes table is in read/write memory, Arm strongly recommends that the table is not updated once the system is running.

Attributes

CNTFID<n> is a 32-bit register.

Field descriptions



Frequency, bits [31:0]

A system counter update frequency, in Hz. Must be an exact divisor of the base frequency. Arm strongly recommends that all frequency values in the Frequency modes table are integer power-of-two divisors of the base frequency.

When the system timer is operating at a lower frequency than the base frequency, the increment applied at each counter update is given by:

increment = (base frequency) / (selected frequency)

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFID<n>

It is IMPLEMENTATION DEFINED whether this register is RO or RW

In a system that supports the Realm Management Extension, the CNTControlBase frame, which includes these registers, is implemented only in the Root physical address space.

In a system that supports Secure and Non-secure physical address spaces, the CNTControlBase frame, which includes these registers, is implemented only in the Secure physical address space.

CNTFID<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTControlBase	`0x020` + (4 * n)	CNTFID<n>

Accesses to this register are **RO or RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTFRQ, Counter-timer Frequency

The CNTFRQ characteristics are:

Purpose

This register is provided so that software can discover the frequency of the system counter. The instance of the register in the CNTCTLBase frame must be programmed with this value as part of system initialization. The value of the register is not interpreted by hardware.

Configuration

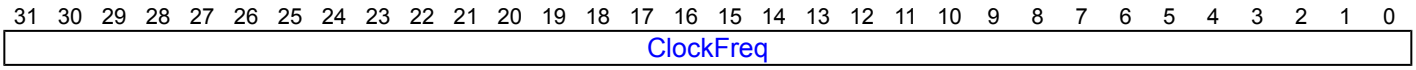
It is IMPLEMENTATION DEFINED whether CNTFRQ is implemented in the Core power domain or in the Debug power domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTFRQ is a 32-bit register.

Field descriptions



ClockFreq, bits [31:0]

Clock frequency. Indicates the system counter clock frequency, in Hz.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFRQ

CNTFRQ must be implemented as an RW register in the CNTCTLBase frame.

In a system that supports the Realm Management Extension, the instance of the register in the CNTCTLBase frame is accessible as follows:

- For Root accesses, it is IMPLEMENTATION DEFINED whether accesses to the register are permitted or behave as RES0.
- For Realm accesses, this register behaves as RES0.

In a system that recognizes two Security states, the instance of the register in the CNTCTLBase frame is only accessible by Secure accesses.

CNTFRQ can be implemented as a RO register in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

For an implemented CNTBaseN frame:

- CNTFRQ is accessible in that frame, as a RO register, if the value of CNTACR<n>.RFRQ is 1.
- Otherwise, the CNTFRQ address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTFRQ is accessible as a RO register in that frame if both:
  - CNTFRQ is accessible in the corresponding CNTBaseN frame.
  - Either the value of CNTEL0ACR.EL0VCTEN is 1 or the value of CNTEL0ACR.EL0PCTEN is 1.
- Otherwise, the CNTFRQ address in that frame is RAZ/WI.

CNTFRQ can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTBaseN	0x010	CNTFRQ

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
Timer	CNTEL0BaseN	0x010	CNTFRQ

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
Timer	CNTCTLBase	0x000	CNTFRQ

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTID, Counter Identification Register

The CNTID characteristics are:

Purpose

Indicates whether counter scaling is implemented.

Configuration

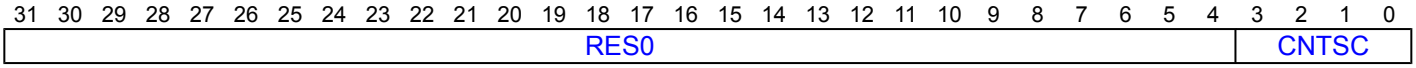
It is IMPLEMENTATION DEFINED whether CNTID is implemented in the Core power domain or in the Debug power domain.

This register is present only when FEAT_CNTSC is implemented. Otherwise, direct accesses to CNTID are RES0.

Attributes

CNTID is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

CNTSC, bits [3:0]

Indicates whether Counter Scaling is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CNTSC	Meaning
0b0000	Counter scaling is not implemented.
0b0001	Counter scaling is implemented.

All other values are reserved.

Access to this field is **RO**.

Accessing CNTID

In a system that supports the Realm Management Extension, the CNTControlBase frame, which includes this register, is implemented only in the Root physical address space.

In a system that supports Secure and Non-secure physical address spaces, the CNTControlBase frame, which includes this register, is implemented only in the Secure physical address space.

CNTID can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTControlBase	0x1C	CNTID

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTNSAR, Counter-timer Non-secure Access Register

The CNTNSAR characteristics are:

Purpose

Provides the highest-level control of whether frames CNTBaseN and CNTEL0BaseN are accessible by Non-secure accesses.

Configuration

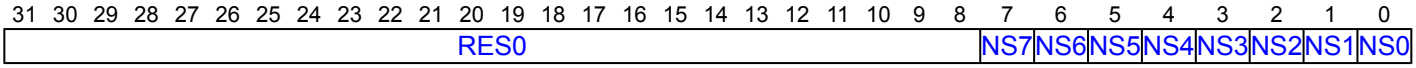
It is IMPLEMENTATION DEFINED whether CNTNSAR is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTNSAR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

NS<n>, bit [n], for n = 7 to 0

Non-secure access to frame n.

NS<n>	Meaning
0b0	Secure access only. Behaves as RES0 to Non-secure accesses. If FEAT_RME is implemented, it is IMPLEMENTATION DEFINED whether Root accesses to the specified registers are permitted or behave as RES0. For Realm accesses, the specified registers behave as RES0.
0b1	Secure and Non-secure accesses permitted. If FEAT_RME is implemented, it is IMPLEMENTATION DEFINED whether Root and Realm accesses to the specified registers are permitted. If not permitted, the specified registers behave as RES0 for Root and Realm accesses.

This bit also determines whether, in the CNTCTLBase frame, CNTACR<n> and CNTVOFF<n> are accessible to Non-secure accesses.

If frame CNTBase<n>:

- Is not implemented, then NS<n> is RES0.
- Is not Configurable access, and is accessible only by Secure accesses, then NS<n> is RES0.
- Is not Configurable access, and is accessible by both Secure and Non-secure accesses, then NS<n> is RES1.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTNSAR

In a system that supports the Realm Management Extension, this register is accessible as follows:

- For Root accesses, it is IMPLEMENTATION DEFINED whether accesses to the register are permitted or behave as RES0.

- For Realm accesses, this register behaves as RES0.

In a system that recognizes two Security states, this register is only accessible by Secure accesses.

CNTNSAR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTCTLBase	0x004	CNTNSAR

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_CTL, Counter-timer Physical Timer Control

The CNTP_CTL characteristics are:

Purpose

Control register for the EL1 physical timer.

Configuration

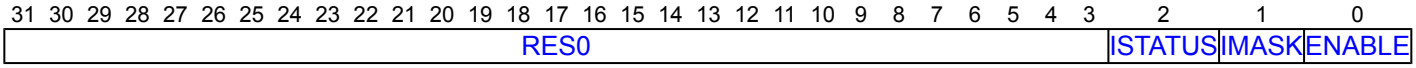
It is IMPLEMENTATION DEFINED whether CNTP_CTL is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTP_CTL is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTP_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CTL

CNTP_CTL can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

For an implemented CNTBaseN frame:

- CNTP_CTL is accessible in that frame if the value of CNTACR<n>.RWPT is 1.
- Otherwise, the CNTP_CTL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTP_CTL is accessible in that frame if both:
  - CNTP_CTL is accessible in the corresponding CNTBaseN frame:
  - The value of CNTEL0ACR.EL0PTEN is 1.
- Otherwise, the CNTP_CTL address in that frame is RAZ/WI.

CNTP_CTL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTBaseN	0x02C	CNTP_CTL

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
Timer	CNTEL0BaseN	0x02C	CNTP_CTL

Accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_CVAL, Counter-timer Physical Timer CompareValue

The CNTP_CVAL characteristics are:

Purpose

Holds the 64-bit compare value for the EL1 physical timer.

Configuration

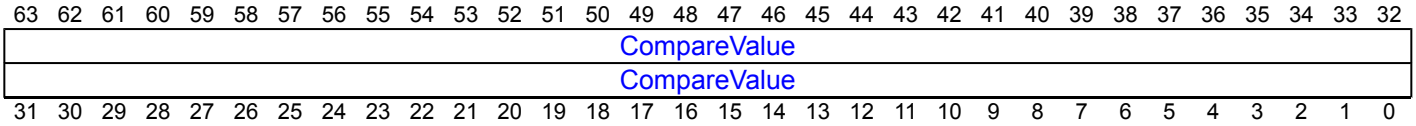
It is IMPLEMENTATION DEFINED whether CNTP_CVAL is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTP_CVAL is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the EL1 physical timer CompareValue.

When CNTP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTP_CTL.ISTATUS is set to 1.
- An interrupt is generated if CNTP_CTL.IMASK is 0.

When CNTP_CTL.ENABLE is 0, the timer condition is not met, but CNTPCT continues to count.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CVAL

CNTP_CVAL can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

For an implemented CNTBaseN frame:

- CNTP_CVAL is accessible in that frame if the value of CNTACR<n>.RWPT is 1.
- Otherwise, the CNTP_CVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTP_CVAL is accessible in that frame if both:
  - CNTP_CVAL is accessible in the corresponding CNTBaseN frame:
  - The value of CNTEL0ACR.EL0PTEN is 1.
- Otherwise, the CNTP_CVAL address in that frame is RAZ/WI.

If the implementation supports 64-bit atomic accesses, then the CNTP_CVAL register must be accessible as an atomic 64-bit value.

CNTP_CVAL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance	Range
Timer	CNTBaseN	0x020	CNTP_CVAL	31:0

Accesses to this register are **RW**.

Component	Frame	Offset	Instance	Range
Timer	CNTBaseN	0x024	CNTP_CVAL	63:32

Accesses to this register are **RW**.

Component	Frame	Offset	Instance	Range
Timer	CNTEL0BaseN	0x020	CNTP_CVAL	31:0

Accesses to this register are **RW**.

Component	Frame	Offset	Instance	Range
Timer	CNTEL0BaseN	0x024	CNTP_CVAL	63:32

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_TVAL, Counter-timer Physical Timer TimerValue

The CNTP_TVAL characteristics are:

Purpose

Holds the timer value for the EL1 physical timer.

Configuration

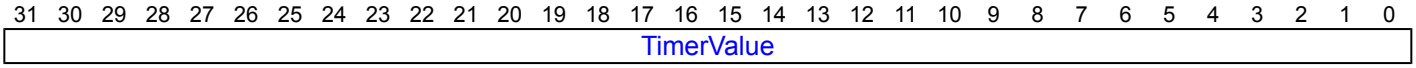
It is IMPLEMENTATION DEFINED whether CNTP_TVAL is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTP_TVAL is a 32-bit register.

Field descriptions



TimerValue, bits [31:0]

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If CNTP_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTP_CTL.ENABLE is 1, the value returned is (CompareValue - CNTPCT).

On a write of this register, CompareValue is set to (CNTPCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CompareValue) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTP_CTL.ISTATUS is set to 1.
- If CNTP_CTL.IMASK is 0, an interrupt is generated.

When CNTP_CTL.ENABLE is 0, the timer condition is not met, but CNTPCT continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_TVAL

CNTP_TVAL can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.

- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

For an implemented CNTBaseN frame:

- CNTP_TVAL is accessible in that frame if the value of CNTACR<n>.RWPT is 1.
- Otherwise, the CNTP_TVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTP_TVAL is accessible in that frame if both:
  - CNTP_TVAL is accessible in the corresponding CNTBaseN frame:
  - The value of CNTEL0ACR.EL0PTEN is 1.
- Otherwise, the CNTP_TVAL address in that frame is RAZ/WI.

CNTP_TVAL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTBaseN	0x028	CNTP_TVAL

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
Timer	CNTEL0BaseN	0x028	CNTP_TVAL

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPCT, Counter-timer Physical Count

The CNTPCT characteristics are:

Purpose

Holds the 64-bit physical count value.

Configuration

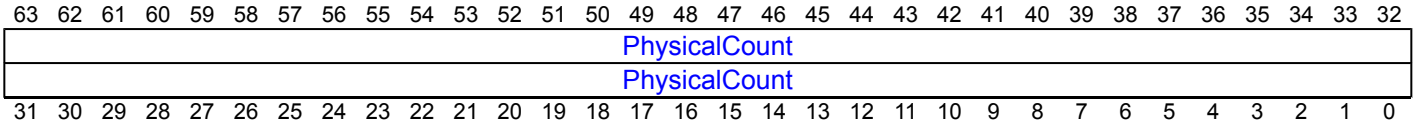
It is IMPLEMENTATION DEFINED whether CNTPCT is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTPCT is a 64-bit register.

Field descriptions



PhysicalCount, bits [63:0]

Physical count value.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCT

CNTPCT can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame, as a RO register.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

For an implemented CNTBaseN frame:

- CNTPCT is accessible in that frame, as a RO register, if the value of CNTACR<n>.RPCT is 1.
- Otherwise, the CNTPCT address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTPCT is accessible in that frame if both:
  - CNTPCT is accessible in the corresponding CNTBaseN frame.
  - The value of CNTEL0ACR.EL0PCTEN is 1.
- Otherwise, the CNTPCT address in that frame is RAZ/WI.

If the implementation supports 64-bit atomic accesses, then the CNTPCT register must be accessible as an atomic 64-bit value.

CNTPCT can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance	Range
Timer	CNTBaseN	0x000	CNTPCT	31:0

Accesses to this register are **RO**.

Component	Frame	Offset	Instance	Range
Timer	CNTBaseN	0x004	CNTPCT	63:32

Accesses to this register are **RO**.

Component	Frame	Offset	Instance	Range
Timer	CNTEL0BaseN	0x000	CNTPCT	31:0

Accesses to this register are **RO**.

Component	Frame	Offset	Instance	Range
Timer	CNTEL0BaseN	0x004	CNTPCT	63:32

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTSCR, Counter Scale Register

The CNTSCR characteristics are:

Purpose

Enables the counter, controls the counter frequency setting, and controls counter behavior during debug.

Configuration

It is IMPLEMENTATION DEFINED whether CNTSCR is implemented in the Core power domain or in the Debug power domain.

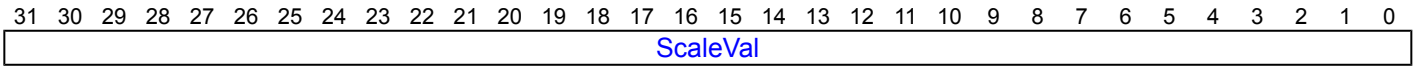
This register is present only when FEAT_CNTSC is implemented. Otherwise, direct accesses to CNTSCR are RES0.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTSCR is a 32-bit register.

Field descriptions



ScaleVal, bits [31:0]

Scale Value

When counter scaling is enabled, ScaleVal is the average amount added to the counter value for one period of the frequency of the Generic counter as described in the CNTFRQ register.

The actual rate of update of the counter value is determined by the counter update frequency.

ScaleVal is expressed as an unsigned fixed point number with an 8-bit integer value and a 24-bit fractional value.

CNTSCR.ScaleVal can only be changed when CNTCR.EN == 0. If the value of this field is changed when CNTCR.EN == 1:

- The counter value becomes UNKNOWN.
- The counter value remains UNKNOWN on future ticks of the clock.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTSCR

In a system that supports the Realm Management Extension, the CNTControlBase frame, which includes this register, is implemented only in the Root physical address space.

In a system that supports Secure and Non-secure physical address spaces, the CNTControlBase frame, which includes this register, is implemented only in the Secure physical address space.

CNTSCR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTControlBase	0x10	CNTSCR

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTSR, Counter Status Register

The CNTSR characteristics are:

Purpose

Provides counter frequency status information.

Configuration

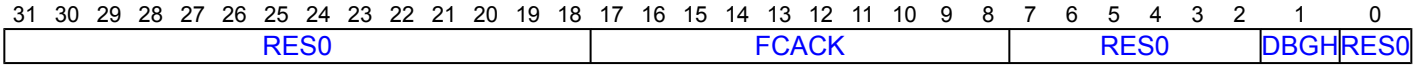
It is IMPLEMENTATION DEFINED whether CNTSR is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTSR is a 32-bit register.

Field descriptions



Bits [31:18]

Reserved, RES0.

FCACK, bits [17:8]

Frequency Change Acknowledge. Indicates the currently selected entry in the Frequency modes table, see 'The Frequency modes table'.

The reset behavior of this field is:

- On a Timer reset, this field resets to `'0000000000'`.

Bits [7:2]

Reserved, RES0.

DBGH, bit [1]

Indicates whether the counter is halted because the Halt-on-debug signal is asserted:

DBGH	Meaning
`0b0`	Counter is not halted.
`0b1`	Counter is halted.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

Accessing CNTSR

In a system that supports the Realm Management Extension, the CNTControlBase frame, which includes this register, is implemented only in the Root physical address space.

In a system that supports Secure and Non-secure physical address spaces, the CNTControlBase frame, which includes this register, is implemented only in the Secure physical address space.

CNTSR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTControlBase	0x004	CNTSR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTTIDR, Counter-timer Timer ID Register

The CNTTIDR characteristics are:

Purpose

Indicates the implemented timers in the physical address space, and their features. For each value of N from 0 to 7 it indicates whether:

- Frame CNTBaseN is a view of an implemented timer.
- Frame CNTBaseN has a second view, CNTEL0BaseN.
- Frame CNTBaseN has a virtual timer capability.

Configuration

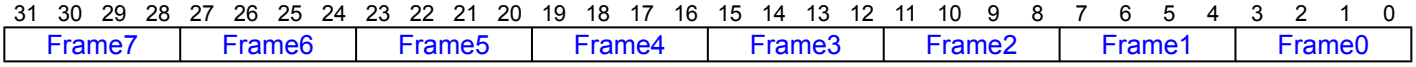
It is IMPLEMENTATION DEFINED whether CNTTIDR is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTTIDR is a 32-bit register.

Field descriptions



Frame<n>, bits [4n+3:4n], for n = 7 to 0

A 4-bit field indicating the features of frame CNTBase<n>.

Bit[3] of the field is RES0.

Bit[2], the FEL0 subfield, indicates whether frame CNTBase<n> has a second view, CNTEL0Base<n>. The possible values of this bit are:

Bit[2]	Meaning
0b0	Frame<n> does not have a second view. The CNTEL0ACR register in the first view of the frame is RES0.
0b1	Frame<n> has a second view, CNTEL0Base<n>.

If bit[0] is 0, bit[2] is RES0.

Bit[1], the FVI subfield, indicates whether both:

- Frame CNTBase<n> implements the virtual timer registers CNTV_CVAL, CNTV_TVAL, and CNTV_CTL.
- This CNTCTLBase frame implements the virtual timer offset register CNTVOFF<n>.

The possible values of bit[1] are:

Bit[1]	Meaning
0b0	Frame<n> does not have virtual capability. The virtual time and offset registers are RES0.
0b1	Frame<n> has virtual capability. The virtual time and offset registers are implemented.

If bit[0] is 0, bit[1] is RES0.

Bit[0], the FI subfield, indicates whether frame CNTBase<n> is implemented. The possible values of this bit are:

Bit[0]	Meaning
0b0	Frame<n> is not implemented. All registers associated with the frame are RES0.
0b1	Frame<n> is implemented.

Accessing CNTTIDR

In a system that supports the Realm Management Extension, it is IMPLEMENTATION DEFINED whether Root and Realm accesses to this register are permitted. If not permitted, this register behaves as RES0 for Root and Realm accesses.

In a system that recognizes two Security states, this register is accessible by both Secure and Non-secure accesses.

CNTTIDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTCTLBase	0x008	CNTTIDR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_CTL, Counter-timer Virtual Timer Control

The CNTV_CTL characteristics are:

Purpose

Control register for the virtual timer.

Configuration

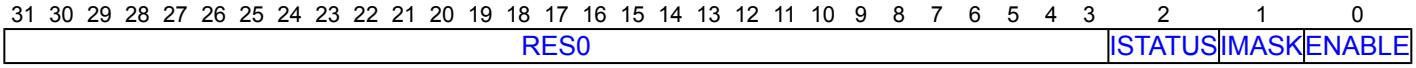
It is IMPLEMENTATION DEFINED whether CNTV_CTL is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTV_CTL is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTV_TVAL continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CTL

CNTV_CTL can be implemented in any implemented CNTBaseN frame that has virtual timer capability, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

- CNTV_CTL is accessible in that frame if the value of CNTACR<n>.RWVT is 1.
- Otherwise, the CNTV_CTL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTV_CTL is accessible in that frame if both:
  - CNTV_CTL is accessible in the corresponding CNTBaseN frame:
  - The value of CNTEL0ACR.EL0VTEN is 1.
- Otherwise, the CNTV_CTL address in that frame is RAZ/WI.

CNTV_CTL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTBaseN	0x03C	CNTV_CTL

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
Timer	CNTEL0BaseN	0x03C	CNTV_CTL

Accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_CVAL, Counter-timer Virtual Timer CompareValue

The CNTV_CVAL characteristics are:

Purpose

Holds the 64-bit compare value for the virtual timer.

Configuration

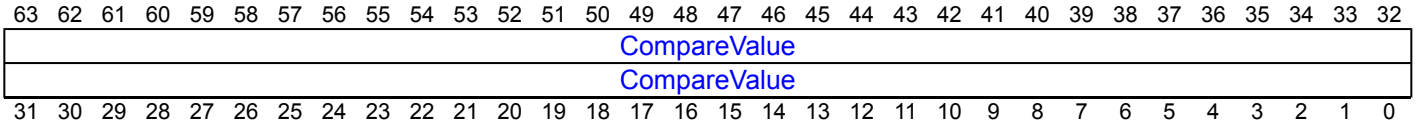
It is IMPLEMENTATION DEFINED whether CNTV_CVAL is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTV_CVAL is a 64-bit register.

Field descriptions



CompareValue, bits [63:0]

Holds the virtual timer CompareValue.

When CNTV_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTV_CTL.ISTATUS is set to 1.
- An interrupt is generated if CNTV_CTL.IMASK is 0.

When CNTV_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CVAL

CNTV_CVAL can be implemented in any implemented CNTBaseN frame that has virtual timer capability, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

- CNTV_CVAL is accessible in that frame if the value of CNTACR<n>.RWVT is 1.

- Otherwise, the CNTV_CVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTV_CVAL is accessible in that frame if both:
  - CNTV_CVAL is accessible in the corresponding CNTBaseN frame:
  - The value of CNTEL0ACR.EL0VTEN is 1.
- Otherwise, the CNTV_CVAL address in that frame is RAZ/WI.

If the implementation supports 64-bit atomic accesses, then the CNTV_CVAL register must be accessible as an atomic 64-bit value.

CNTV_CVAL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance	Range
Timer	CNTBaseN	0x030	CNTV_CVAL	31:0

Accesses to this register are **RW**.

Component	Frame	Offset	Instance	Range
Timer	CNTBaseN	0x034	CNTV_CVAL	63:32

Accesses to this register are **RW**.

Component	Frame	Offset	Instance	Range
Timer	CNTEL0BaseN	0x030	CNTV_CVAL	31:0

Accesses to this register are **RW**.

Component	Frame	Offset	Instance	Range
Timer	CNTEL0BaseN	0x034	CNTV_CVAL	63:32

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_TVAL, Counter-timer Virtual Timer TimerValue

The CNTV_TVAL characteristics are:

Purpose

Holds the timer value for the virtual timer.

Configuration

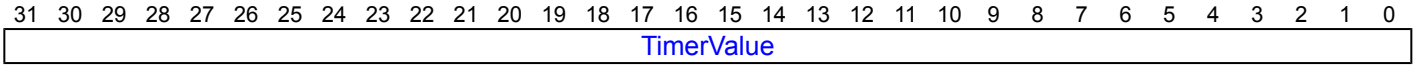
It is IMPLEMENTATION DEFINED whether CNTV_TVAL is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTV_TVAL is a 32-bit register.

Field descriptions



TimerValue, bits [31:0]

The TimerValue view of the virtual timer.

On a read of this register:

- If CNTV_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTV_CTL.ENABLE is 1, the value returned is (CompareValue - CNTVCT).

On a write of this register, CompareValue is set to (CNTVCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTV_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTV_CTL.ISTATUS is set to 1.
- If CNTV_CTL.IMASK is 0, an interrupt is generated.

When CNTV_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_TVAL

CNTV_TVAL can be implemented in any implemented CNTBaseN frame that has virtual timer capability, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.

- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

- CNTV_TVAL is accessible in that frame if the value of CNTACR<n>.RWVT is 1.
- Otherwise, the CNTV_TVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTV_TVAL is accessible in that frame if both:
  - CNTV_TVAL is accessible in the corresponding CNTBaseN frame.
  - The value of CNTEL0ACR.EL0VTEN is 1.
- Otherwise, the CNTV_TVAL address in that frame is RAZ/WI.

CNTV_TVAL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTBaseN	0x038	CNTV_TVAL

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
Timer	CNTEL0BaseN	0x038	CNTV_TVAL

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVCT, Counter-timer Virtual Count

The CNTVCT characteristics are:

Purpose

Holds the 64-bit virtual count value.

Configuration

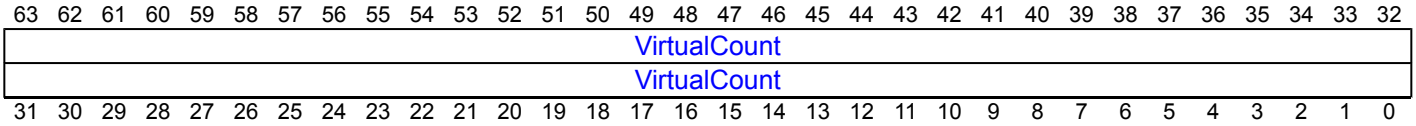
It is IMPLEMENTATION DEFINED whether CNTVCT is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTVCT is a 64-bit register.

Field descriptions



VirtualCount, bits [63:0]

Virtual count value.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCT

CNTVCT can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame, as a RO register.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

For an implemented CNTBaseN frame:

- CNTVCT is accessible in that frame, as a RO register, if the value of CNTACR<n>.RVCT is 1.
- Otherwise, the CNTVCT address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTVCT is accessible in that frame if both:
  - CNTVCT is accessible in the corresponding CNTBaseN frame.
  - The value of CNTEL0ACR.EL0VCTEN is 1.
- Otherwise, the CNTVCT address in that frame is RAZ/WI.

If the implementation supports 64-bit atomic accesses, then the CNTVCT register must be accessible as an atomic 64-bit value.

CNTVCT can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance	Range
Timer	CNTBaseN	0x008	CNTVCT	31:0

Accesses to this register are **RO**.

Component	Frame	Offset	Instance	Range
Timer	CNTBaseN	0x00C	CNTVCT	63:32

Accesses to this register are **RO**.

Component	Frame	Offset	Instance	Range
Timer	CNTEL0BaseN	0x008	CNTVCT	31:0

Accesses to this register are **RO**.

Component	Frame	Offset	Instance	Range
Timer	CNTEL0BaseN	0x00C	CNTVCT	63:32

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVOFF, Counter-timer Virtual Offset

The CNTVOFF characteristics are:

Purpose

Holds the 64-bit virtual offset for a CNTBaseN frame that has virtual timer capability. This is the offset between real time and virtual time.

Configuration

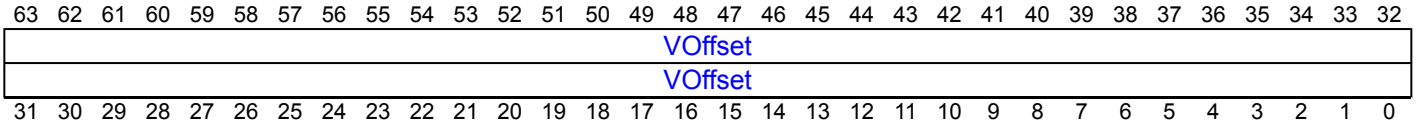
It is IMPLEMENTATION DEFINED whether CNTVOFF is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTVOFF is a 64-bit register.

Field descriptions



VOffset, bits [63:0]

Virtual offset.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVOFF

CNTVOFF is implemented, as a RO register, in any implemented CNTBaseN frame that has virtual timer capability.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

- CNTVOFF is accessible in that frame, as a RO register, if the value of CNTACR<n>.RVOFF is 1.
- Otherwise, the CNTVOFF address in that frame is RAZ/WI.

Note

CNTVOFF is never visible in any CNTEL0BaseN frame. This means that the CNTVOFF address in any implemented CNTEL0BaseN frame is RAZ/WI.

In an implementation that supports 64-bit atomic accesses, then the CNTVOFF register must be accessible as an atomic 64-bit value.

CNTVOFF can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Range
Timer	CNTBaseN	0x018	31:0

Accesses to this register are **RO**.

Component	Frame	Offset	Range
Timer	CNTBaseN	0x01C	63:32

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVOFF<n>, Counter-timer Virtual Offsets, n = 0 - 7

The CNTVOFF<n> characteristics are:

Purpose

Holds the 64-bit virtual offset for frame CNTBase<n>. This is the offset between real time and virtual time.

Configuration

It is IMPLEMENTATION DEFINED whether CNTVOFF<n> is implemented in the Core power domain or in the Debug power domain.

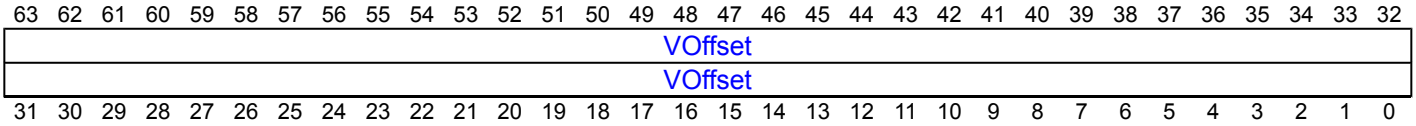
Implementation of this register is OPTIONAL.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTVOFF<n> is a 64-bit register.

Field descriptions



VOffset, bits [63:0]

Virtual offset.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVOFF<n>

In the CNTCTLBase frame a CNTVOFF<n> register must be implemented, as a RW register, for each CNTBaseN frame that has virtual timer capability. For more information, see 'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames'.

Note

The value of <n> in an instance of CNTVOFF<n> specifies the value of N for the associated CNTBaseN frame.

In a system that supports the Realm Management Extension, CNTNSAR.NS<n> describes how these registers can be accessed by Root or Realm accesses.

In a system that recognizes two Security states, for any CNTVOFF<n> register in the CNTCTLBase frame:

- CNTVOFF<n> is always accessible by Secure accesses.
- CNTNSAR.NS<n> determines whether CNTVOFF<n> is accessible by Non-secure accesses.

The register location of any unimplemented CNTVOFF<n> register in the CNTCTLBase frame is RAZ/WI.

The CNTVOFF<n> register is accessible in the CNTBaseN frame using CNTVOFF.

In an implementation that supports 64-bit atomic accesses, then the CNTVOFF<n> registers must be accessible as atomic 64-bit values.

CNTVOFF<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Range
Timer	CNTCTLBase	`0x080` + (8 * n)	31:0

Accesses to this register are **RW**.

Component	Frame	Offset	Range
Timer	CNTCTLBase	`0x084` + (8 * n)	63:32

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CounterID<n>, Counter ID registers, n = 0 - 11

The CounterID<n> characteristics are:

Purpose

IMPLEMENTATION DEFINED identification registers 0 to 11 for the memory-mapped Generic Timer.

Configuration

It is IMPLEMENTATION DEFINED whether CounterID<n> is implemented in the Core power domain or in the Debug power domain.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

These registers are implemented independently in each of the implemented Generic Timer memory-mapped frames.

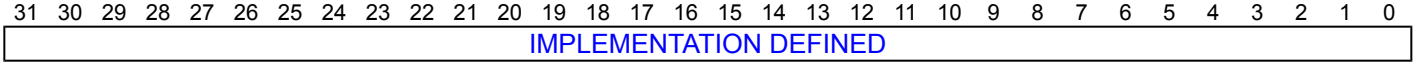
If the implementation of the Counter ID registers requires an architecture version, the value for this version of the Arm Generic Timer is version 0.

The Counter ID registers can be implemented as a set of CoreSight ID registers, comprising Peripheral ID Registers and Component ID Registers. An implementation of these registers for the Generic Timer must use a Component class value of `0xF`.

Attributes

CounterID<n> is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing CounterID<n>

These registers must be implemented, as RO registers, in every implemented Generic Timer memory-mapped frame.

For the CNTCTLBase frame, in a system that supports the Realm Management Extension, it is IMPLEMENTATION DEFINED whether Root and Realm accesses to these registers are permitted. If not permitted, these registers behave as RES0 for Root and Realm accesses.

For the CNTCTLBase frame, in a system that recognizes two Security states these registers are accessible by both Secure and Non-secure accesses.

For the CNTControlBase frame, in a system that supports the Realm Management Extension, the frame is implemented only in the Root physical address space, meaning these registers are implemented only in the Root physical address space.

For the CNTControlBase frame, in a system that supports Secure and Non-secure physical address spaces, the frame is implemented only in the Secure physical address space, meaning these registers are implemented only in the Secure physical address space.

For the CNTReadBase frame, these registers are accessible in all physical address spaces.

For the CNTBaseN frames, 'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that supports the Realm Management Extension, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Root and Realm accesses.

- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses. The CNTBaseN frame is always accessible by Secure accesses.

CounterID<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
Timer	CNTControlBase	`0xFD0` + (4 * n)	CounterID<n>

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
Timer	CNTReadBase	`0xFD0` + (4 * n)	CounterID<n>

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
Timer	CNTBaseN	`0xFD0` + (4 * n)	CounterID<n>

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
Timer	CNTEL0BaseN	`0xFD0` + (4 * n)	CounterID<n>

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
Timer	CNTCTLBase	`0xFD0` + (4 * n)	CounterID<n>

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIAPPCLEAR, CTI Application Trigger Clear register

The CTIAPPCLEAR characteristics are:

Purpose

Clears the application triggers.

Configuration

External register CTIAPPCLEAR bits [31:0] are architecturally mapped to External register CTIAPPSET[31:0].

CTIAPPCLEAR is in the Debug power domain.

Attributes

CTIAPPCLEAR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23
APPCLEAR31	APPCLEAR30	APPCLEAR29	APPCLEAR28	APPCLEAR27	APPCLEAR26	APPCLEAR25	APPCLEAR24	APPCLEAR

APPCLEAR<x>, bit [x], for x = 31 to 0

Application trigger <x> disable.

Writing to this bit has the following effect:

APPCLEAR<x>	Meaning
0b0	No effect.
0b1	Clear corresponding application trigger to 0 and clear the corresponding channel event.

If the ECT does not support multicycle channel events, use of CTIAPPCLEAR is deprecated and the debugger must only use CTIAPPPULSE.

Accessing this field has the following behavior:

- When x >= UInt(CTIDEVID.NUMCHAN), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RAZ/W1C**.

Accessing CTIAPPCLEAR

CTIAPPCLEAR can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x018	CTIAPPCLEAR

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **WI**.
- Otherwise, accesses to this register are **WO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIAPPPULSE, CTI Application Pulse register

The CTIAPPPULSE characteristics are:

Purpose

Causes event pulses to be generated on ECT channels.

Configuration

CTIAPPPULSE is in the Debug power domain.

Attributes

CTIAPPPULSE is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23
APPPULSE31	APPPULSE30	APPPULSE29	APPPULSE28	APPPULSE27	APPPULSE26	APPPULSE25	APPPULSE24	APPPULSE2

APPPULSE<x>, bit [x], for x = 31 to 0

Generate event pulse on ECT channel <x>.

APPPULSE<x>	Meaning
0b0	No effect.
0b1	Channel <x> event pulse generated.

Note

- The CTIAPPPULSE operation does not affect the state of the application trigger. If the channel is active, either because of an earlier event or from the application trigger, then the value written to CTIAPPPULSE might have no effect.
- Multiple pulse events that occur close together might be merged into a single pulse event.

Accessing this field has the following behavior:

- When x >= UInt(CTIDEVID.NUMCHAN), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **WO/UNKNOWN**.

Accessing CTIAPPPULSE

It is CONSTRAINED UNPREDICTABLE whether a write to CTIAPPPULSE generates an event on a channel if CTICONTROL.GLBEN is 0.

CTIAPPPULSE can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x01C	CTIAPPPULSE

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **WI**.
- Otherwise, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIAPPSET, CTI Application Trigger Set register

The CTIAPPSET characteristics are:

Purpose

Sets the application triggers.

Configuration

External register CTIAPPSET bits [31:0] are architecturally mapped to External register CTIAPPCLEAR[31:0].

CTIAPPSET is in the Debug power domain.

Attributes

CTIAPPSET is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	
APPSET31	APPSET30	APPSET29	APPSET28	APPSET27	APPSET26	APPSET25	APPSET24	APPSET23	APPSET22	APPSET21	AP

APPSET<x>, bit [x], for x = 31 to 0

Application trigger <x> enable.

APPSET<x>	Meaning
0b0	Reading this means the application trigger is inactive. Writing this has no effect.
0b1	Reading this means the application trigger is active. Writing this sets the corresponding application trigger to 1 and generates a channel event.

If the ECT does not support multicycle channel events, use of CTIAPPSET is deprecated and the debugger must only use CTIAPPPULSE.

The reset behavior of this field is:

- On an External debug reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When x >= UInt(CTIDEVID.NUMCHAN), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing CTIAPPSET

CTIAPPSET can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x014	CTIAPPSET

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIAUTHSTATUS, CTI Authentication Status register

The CTIAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for CTI.

Configuration

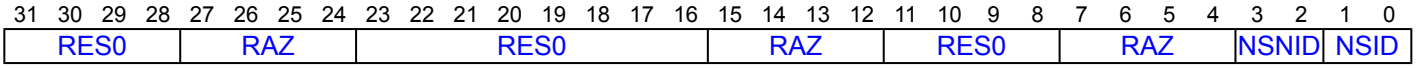
CTIAUTHSTATUS is in the Debug power domain.

This register is OPTIONAL, and is required for CoreSight compliance.

Attributes

CTIAUTHSTATUS is a 32-bit register.

Field descriptions



Bits [31:28]

Reserved, RES0.

Bits [27:24]

Reserved, RAZ.

Bits [23:16]

Reserved, RES0.

Bits [15:12]

Reserved, RAZ.

Bits [11:8]

Reserved, RES0.

Bits [7:4]

Reserved, RAZ.

NSNID, bits [3:2]

If EL3 is implemented, this field holds the same value as DBGAUTHSTATUS_EL1.NSNID.

If EL3 is not implemented and the implemented Security state is Secure state, this field holds the same value as DBGAUTHSTATUS_EL1.SNID.

NSID, bits [1:0]

If EL3 is implemented, this field holds the same value as DBGAUTHSTATUS_EL1.NSID.

If EL3 is not implemented and the implemented Security state is Secure state, this field holds the same value as DBGAUTHSTATUS_EL1.SID.

Accessing CTIAUTHSTATUS

CTIAUTHSTATUS can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFB8	CTIAUTHSTATUS

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICHINSTATUS, CTI Channel In Status register

The CTICHINSTATUS characteristics are:

Purpose

Provides the raw status of the ECT channel inputs to the CTI.

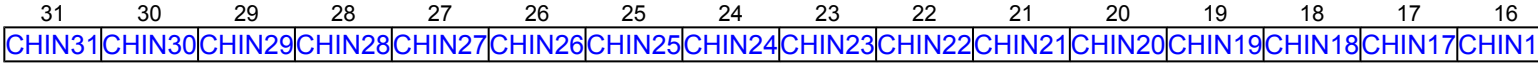
Configuration

CTICHINSTATUS is in the Debug power domain.

Attributes

CTICHINSTATUS is a 32-bit register.

Field descriptions



CHIN<n>, bit [n], for n = 31 to 0

Input channel <n> status.

CHIN<n>	Meaning
0b0	Input channel <n> is inactive.
0b1	Input channel <n> is active.

If the ECT channels do not support multicycle events then it is IMPLEMENTATION DEFINED whether an input channel can be observed as active.

Accessing this field has the following behavior:

- When n >= UInt(CTIDEVID.NUMCHAN), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RO**.

Accessing CTICHINSTATUS

CTICHINSTATUS can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x138	CTICHINSTATUS

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICHOUTSTATUS, CTI Channel Out Status register

The CTICHOUTSTATUS characteristics are:

Purpose

Provides the status of the ECT channel outputs from the CTI.

Configuration

CTICHOUTSTATUS is in the Debug power domain.

Attributes

CTICHOUTSTATUS is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20
CHOUT31	CHOUT30	CHOUT29	CHOUT28	CHOUT27	CHOUT26	CHOUT25	CHOUT24	CHOUT23	CHOUT22	CHOUT21	CHOUT20	C

CHOUT<n>, bit [n], for n = 31 to 0

Output channel <n> status.

CHOUT<n>	Meaning
0b0	Output channel <n> is inactive.
0b1	Output channel <n> is active.

If the ECT channels do not support multicycle events then it is IMPLEMENTATION DEFINED whether an output channel can be observed as active.

Note

The value in CTICHOUTSTATUS is after gating by the channel gate. For more information, see CTIGATE.

Accessing this field has the following behavior:

- When n >= UInt(CTIDEVID.NUMCHAN), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RO**.

Accessing CTICHOUTSTATUS

CTICHOUTSTATUS can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x13C	CTICHOUTSTATUS

Accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICIDR0, CTI Component Identification Register 0

The CTICIDR0 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Component Identification scheme'.

Configuration

CTICIDR0 is in the Debug power domain.

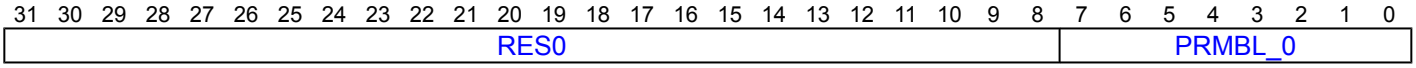
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTICIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Preamble.

Reads as `0x0D`.

Access to this field is **RO**.

Accessing CTICIDR0

CTICIDR0 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	`0xFF0`	CTICIDR0

Accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICIDR1, CTI Component Identification Register 1

The CTICIDR1 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Component Identification scheme'.

Configuration

CTICIDR1 is in the Debug power domain.

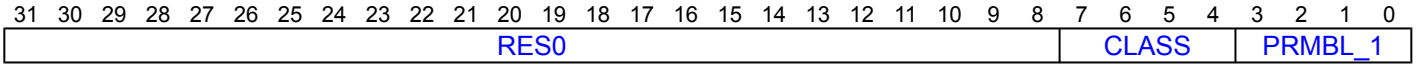
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTICIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

CLASS	Meaning
0b1001	CoreSight component.

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

Access to this field is **RO**.

PRMBL_1, bits [3:0]

Preamble.

Reads as 0b0000.

Access to this field is **RO**.

Accessing CTICIDR1

CTICIDR1 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFF4	CTICIDR1

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICIDR2, CTI Component Identification Register 2

The CTICIDR2 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Component Identification scheme'.

Configuration

CTICIDR2 is in the Debug power domain.

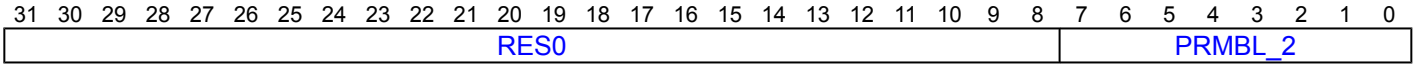
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTICIDR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Preamble.

Reads as `0x05`.

Access to this field is **RO**.

Accessing CTICIDR2

CTICIDR2 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	`0xFF8`	CTICIDR2

Accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICIDR3, CTI Component Identification Register 3

The CTICIDR3 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Component Identification scheme'.

Configuration

CTICIDR3 is in the Debug power domain.

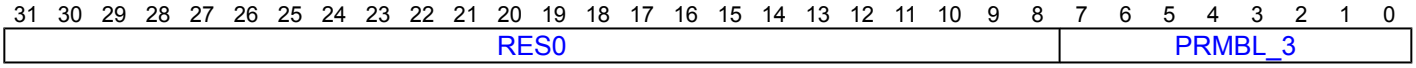
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTICIDR3 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Preamble.

Reads as `0xB1`.

Access to this field is **RO**.

Accessing CTICIDR3

CTICIDR3 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	`0xFFC`	CTICIDR3

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICLAIMCLR, CTI CLAIM Tag Clear register

The CTICLAIMCLR characteristics are:

Purpose

Used by software to read the values of the CLAIM bits, and to clear CLAIM tag bits to 0.

Configuration

External register CTICLAIMCLR bits [31:0] are architecturally mapped to External register CTICLAIMSET[31:0].

CTICLAIMCLR is in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

CTICLAIMCLR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18
CLAIM31	CLAIM30	CLAIM29	CLAIM28	CLAIM27	CLAIM26	CLAIM25	CLAIM24	CLAIM23	CLAIM22	CLAIM21	CLAIM20	CLAIM19	CLAI

CLAIM<m>, bit [m], for m = 31 to 0

Claim Tag Clear. Indicates the current status of Claim Tag bit <m>, and is used to clear Claim Tag bit <m> to 0.

CLAIM<m>	Meaning
0b0	On a read: Claim Tag bit <m> is not set. On a write: Ignored.
0b1	On a read: Claim Tag bit <m> is set. On a write: Clear Claim tag bit <m> to 0.

The number of Claim Tag bits implemented is indicated in CTICLAIMSET.

An External Debug reset clears the CLAIM tag bits to 0.

Accessing this field has the following behavior:

- When m >= NUM_CTI_CLAIM_TAGS, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing CTICLAIMCLR

CTICLAIMCLR can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFA4	CTICLAIMCLR

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICLAIMSET, CTI CLAIM Tag Set register

The CTICLAIMSET characteristics are:

Purpose

Used by software to set CLAIM bits to 1.

Configuration

External register CTICLAIMSET bits [31:0] are architecturally mapped to External register CTICLAIMCLR[31:0].

CTICLAIMSET is in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

CTICLAIMSET is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18
CLAIM31	CLAIM30	CLAIM29	CLAIM28	CLAIM27	CLAIM26	CLAIM25	CLAIM24	CLAIM23	CLAIM22	CLAIM21	CLAIM20	CLAIM19	CLAI

CLAIM<m>, bit [m], for m = 31 to 0

Claim Tag Set. Indicates whether Claim Tag bit <m> is implemented, and is used to set Claim Tag bit <m> to 1.

CLAIM<m>	Meaning
0b0	On a read: Claim Tag bit <m> is not implemented. On a write: Ignored.
0b1	On a read: Claim Tag bit <m> is implemented. On a write: Set Claim Tag bit <m> to 1.

An External Debug reset clears the CLAIM tag bits to 0.

Accessing this field has the following behavior:

- When m >= NUM_CTI_CLAIM_TAGS, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RAO/W1S**.

Accessing CTICLAIMSET

CTICLAIMSET can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFA0	CTICLAIMSET

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICONTROL, CTI Control register

The CTICONTROL characteristics are:

Purpose

Controls whether the CTI is enabled.

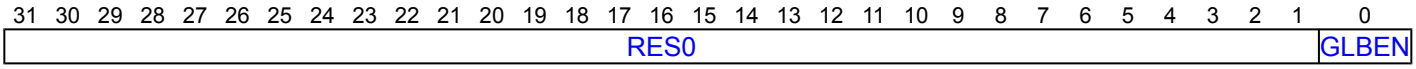
Configuration

CTICONTROL is in the Debug power domain.

Attributes

CTICONTROL is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

GLBEN, bit [0]

Enables or disables the CTI mapping functions. Possible values of this field are:

GLBEN	Meaning
0b0	CTI mapping functions and application trigger disabled.
0b1	CTI mapping functions and application trigger enabled.

When GLBEN is 0, the input channel to output trigger, input trigger to output channel, and application trigger functions are disabled and do not signal new events on either output triggers or output channels. If a previously asserted output trigger has not been acknowledged, it is CONSTRAINED UNPREDICTABLE which of the following occurs:

- The output trigger remains asserted after the mapping functions are disabled.
- The output trigger is deasserted after the mapping functions are disabled.

All output triggers are disabled by CTI reset.

If the ECT supports multicycle channel events any existing output channel events will be terminated.

The reset behavior of this field is:

- On an External debug reset, this field resets to '0'.

Accessing CTICONTROL

CTICONTROL can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x000	CTICONTROL

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVAFF0, CTI Device Affinity register 0

The CTIDEVAFF0 characteristics are:

Purpose

Copy of the low half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the CTI component relates to.

Configuration

CTIDEVAFF0 is in the Debug power domain.

If the CTI is CTIv1, this register is OPTIONAL. If the CTI is CTIv2, this register is mandatory.

Arm recommends that the CTI is CTIv2.

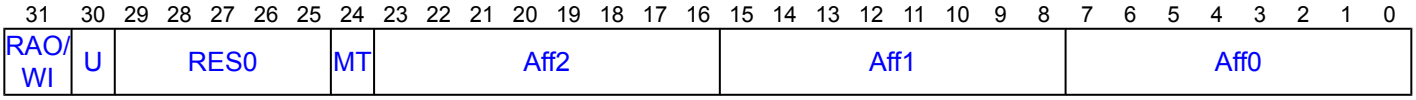
In an Armv8.5 compliant implementation, the CTI must be CTIv2.

If this register is implemented, then CTIDEVAFF1 must also be implemented. If the CTI of a PE does not implement the CTI Device Affinity registers, the CTI block of the external debug memory map must be located 64KB above the debug registers in the external debug interface.

Attributes

CTIDEVAFF0 is a 32-bit register.

Field descriptions



Bit [31]

Reserved, RAO/WI.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Access to this field is **RO**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is **RO**.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIDEVAFF0

CTIDEVAFF0 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFA8	CTIDEVAFF0

Accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVAFF1, CTI Device Affinity register 1

The CTIDEVAFF1 characteristics are:

Purpose

Copy of the high half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the CTI component relates to.

Configuration

CTIDEVAFF1 is in the Debug power domain.

If the CTI is CTIv1, this register is OPTIONAL. If the CTI is CTIv2, this register is mandatory.

Arm recommends that the CTI is CTIv2.

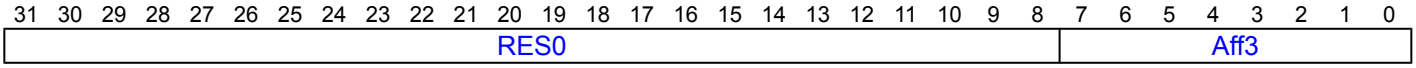
In an Armv8.5 compliant implementation, the CTI must be CTIv2.

If this register is implemented, then CTIDEVAFF0 must also be implemented. If the CTI of a PE does not implement the CTI Device Affinity registers, the CTI block of the external debug memory map must be located 64KB above the debug registers in the external debug interface.

Attributes

CTIDEVAFF1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

Aff3, bits [7:0]

Affinity level 3. See the description of CTIDEVAFF0.Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIDEVAFF1

CTIDEVAFF1 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFAC	CTIDEVAFF1

Accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVARCH, CTI Device Architecture register

The CTIDEVARCH characteristics are:

Purpose

Identifies the programmers' model architecture of the CTI component.

Configuration

CTIDEVARCH is in the Debug power domain.

If the CTI is CTIv1, this register is OPTIONAL. If the CTI is CTIv2, this register is mandatory.

Arm recommends that the CTI is CTIv2.

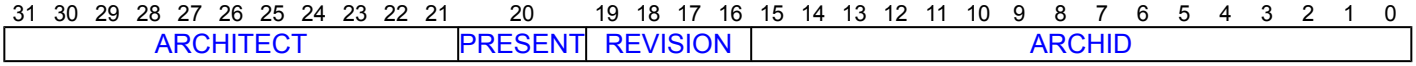
In an Armv8.5 compliant implementation, the CTI must be CTIv2.

If this register is not implemented, CTIDEVAFF0 and CTIDEVAFF1 are also not implemented.

Attributes

CTIDEVARCH is a 32-bit register.

Field descriptions



ARCHITECT, bits [31:21]

Defines the architect of the component. For CTI, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, `0b0100`.

Bits [27:21] are the JEP106 identification code, `0b0111011`.

Reads as `0b01000111011`.

Access to this field is **RO**.

PRESENT, bit [20]

DEVARCH present. Indicates that the CTIDEVARCH register is present.

Reads as `0b1`.

Access to this field is **RO**.

REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

REVISION	Meaning
`0b0000`	First revision.
`0b0001`	As `0b0000`, and also adds support for CTIDEVCTL.

All other values are reserved.

When FEAT_DoPD is implemented, the value `0b0000` is not permitted.

Access to this field is **RO**.

ARCHID, bits [15:0]

Defines this part to be an Armv8 debug component. For architectures defined by Arm this is further subdivided.

For CTI:

- Bits [15:12] are the architecture version, `0x1`.
- Bits [11:0] are the architecture part number, `0xA14`.

This corresponds to CTI architecture version CTIv2.

Reads as `0x1A14`.

Access to this field is **RO**.

Accessing CTIDEVARCH

CTIDEVARCH can be accessed through the external debug interface:

Component	Offset	Instance
CTI	`0xFBC`	CTIDEVARCH

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVCTL, CTI Device Control register

The CTIDEVCTL characteristics are:

Purpose

Provides target-specific device controls

Configuration

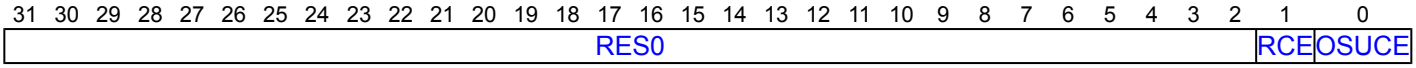
CTIDEVCTL is in the Debug power domain.

This register is present only when FEAT_DoPD is implemented. Otherwise, direct accesses to CTIDEVCTL are RES0.

Attributes

CTIDEVCTL is a 32-bit register.

Field descriptions



Bits [31:2]

Reserved, RES0.

RCE, bit [1]

Reset Catch Enable.

RCE	Meaning
0b0	Reset Catch debug event disabled.
0b1	Reset Catch debug event enabled.

The reset behavior of this field is:

- On an External debug reset, this field resets to '0'.

OSUCE, bit [0]

OS Unlock Catch Enable

OSUCE	Meaning
0b0	OS Unlock Catch debug event disabled.
0b1	OS Unlock Catch debug event enabled.

The reset behavior of this field is:

- On an External debug reset, this field resets to '0'.

Accessing CTIDEVCTL

CTIDEVCTL can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x150	CTIDEVCTL

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVID, CTI Device ID register 0

The CTIDEVID characteristics are:

Purpose

Describes the CTI component to the debugger.

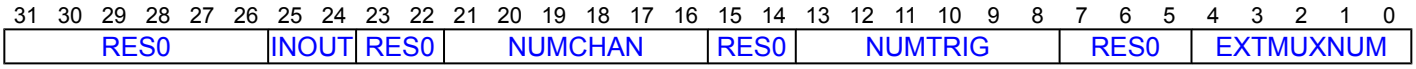
Configuration

CTIDEVID is in the Debug power domain.

Attributes

CTIDEVID is a 32-bit register.

Field descriptions



Bits [31:26]

Reserved, RES0.

INOUT, bits [25:24]

Input/output options. Indicates presence of the input gate. If the CTM is not implemented or CTIv2 is not implemented, this field is RAZ.

The value of this field is an IMPLEMENTATION DEFINED choice of:

INOUT	Meaning
0b00	CTIGATE does not mask propagation of input events from external channels.
0b01	CTIGATE masks propagation of input events from external channels.

All other values are reserved.

Access to this field is **RO**.

Bits [23:22]

Reserved, RES0.

NUMCHAN, bits [21:16]

Number of ECT channels implemented. For Armv8, valid values are:

- 0b000011 3 channels (0..2) implemented.
- 0b000100 4 channels (0..3) implemented.
- 0b000101 5 channels (0..4) implemented.
- 0b000110 6 channels (0..5) implemented.

and so on up to 0b100000, 32 channels (0..31) implemented.

All other values are reserved.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [15:14]

Reserved, RES0.

NUMTRIG, bits [13:8]

Upper bound for number of triggers. The indices of all implemented input and output triggers are less than this value.

There is no guarantee that all of the input and output triggers, including the highest numbered, are connected to any components, or that the implementation of input and output triggers is symmetrical.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [7:5]

Reserved, RES0.

EXTMUXNUM, bits [4:0]

Number of multiplexors available on triggers. This value is used in conjunction with External Control register, ASICCTL.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIDEVID

CTIDEVID can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFC8	CTIDEVID

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVID1, CTI Device ID register 1

The CTIDEVID1 characteristics are:

Purpose

Reserved for future information about the CTI component to the debugger.

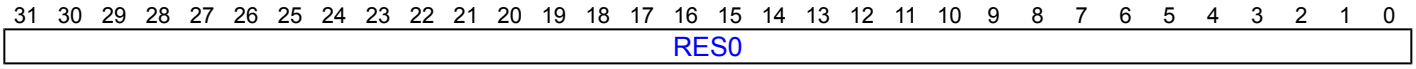
Configuration

CTIDEVID1 is in the Debug power domain.

Attributes

CTIDEVID1 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing CTIDEVID1

CTIDEVID1 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFC4	CTIDEVID1

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVID2, CTI Device ID register 2

The CTIDEVID2 characteristics are:

Purpose

Reserved for future information about the CTI component to the debugger.

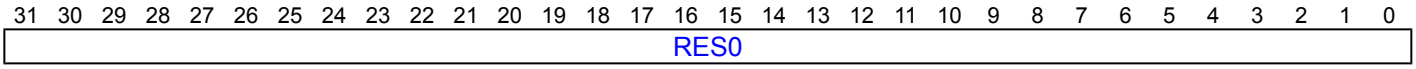
Configuration

CTIDEVID2 is in the Debug power domain.

Attributes

CTIDEVID2 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing CTIDEVID2

CTIDEVID2 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFC0	CTIDEVID2

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVTYPE, CTI Device Type register

The CTIDEVTYPE characteristics are:

Purpose

Indicates to a debugger that this component is part of a PE's cross-trigger interface.

Configuration

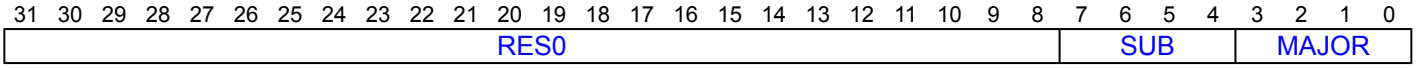
CTIDEVTYPE is in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

CTIDEVTYPE is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Subtype. Indicates this is a component within a PE.

Reads as `0b0001`.

Access to this field is **RO**.

MAJOR, bits [3:0]

Major type. Indicates this is a cross-trigger component.

Reads as `0b0100`.

Access to this field is **RO**.

Accessing CTIDEVTYPE

CTIDEVTYPE can be accessed through the external debug interface:

Component	Offset	Instance
CTI	`0xFCC`	CTIDEVTYPE

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIGATE, CTI Channel Gate Enable register

The CTIGATE characteristics are:

Purpose

Determines whether events on channels propagate through the CTM to other ECT components, or from the CTM into the CTI.

Configuration

CTIGATE is in the Debug power domain.

Attributes

CTIGATE is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
GATE31	GATE30	GATE29	GATE28	GATE27	GATE26	GATE25	GATE24	GATE23	GATE22	GATE21	GATE20	GATE19	GATE18	GATE17

GATE<x>, bit [x], for x = 31 to 0

Channel <x> gate enable.

GATE<x>	Meaning
0b0	Disable output and, if CTIDEVID.INOUT == 0b01, input channel <x> propagation.
0b1	Enable output and, if CTIDEVID.INOUT == 0b01, input channel <x> propagation.

If GATE<x> is set to 0, no new events will be propagated to the ECT, and if the ECT supports multicycle channel events any existing output channel events will be terminated.

The reset behavior of this field is:

- On an External debug reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When x >= UInt(CTIDEVID.NUMCHAN), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RW**.

Accessing CTIGATE

CTIGATE can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x140	CTIGATE

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIINEN<n>, CTI Input Trigger to Output Channel Enable registers, n = 0 - 31

The CTIINEN<n> characteristics are:

Purpose

Enables the signaling of an event on output channels when input trigger event n is received by the CTI.

Configuration

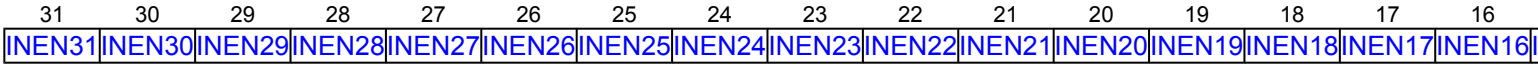
CTIINEN<n> is in the Debug power domain.

If input trigger n is not implemented or not connected, CTIINEN<n> is RES0.

Attributes

CTIINEN<n> is a 32-bit register.

Field descriptions



INEN<x>, bit [x], for x = 31 to 0

Input trigger <n> to output channel <x> enable.

INEN<x>	Meaning
0b0	Input trigger <n> will not generate an event on output channel <x>.
0b1	Input trigger <n> will generate an event on output channel <x>.

The reset behavior of this field is:

- On an External debug reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When x >= UInt(CTIDEVID.NUMCHAN), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RW**.

Accessing CTIINEN<n>

CTIINEN<n> can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x020 + (4 * n)	CTIINEN<n>

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIINTACK, CTI Output Trigger Acknowledge register

The CTIINTACK characteristics are:

Purpose

Can be used to deactivate the output triggers.

Configuration

CTIINTACK is in the Debug power domain.

Attributes

CTIINTACK is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14
ACK31	ACK30	ACK29	ACK28	ACK27	ACK26	ACK25	ACK24	ACK23	ACK22	ACK21	ACK20	ACK19	ACK18	ACK17	ACK16	ACK15	ACK1

ACK<n>, bit [n], for n = 31 to 0

Acknowledge for output trigger <n>.

If any of the following is true, writes to ACK<n> are ignored:

- n >= CTIDEVID.NUMTRIG, the number of implemented triggers.
- Output trigger n is not active.
- The channel mapping function output, as controlled by CTIOUTEN<n>, is still active.

Otherwise, if any of the following are true, ACK<n> is RES0:

- Output trigger n is not implemented.
- Output trigger n is not connected.
- Output trigger n is self-acknowledging and does not require software acknowledge.

Otherwise, the behavior on writes to ACK<n> is as follows:

ACK<n>	Meaning
0b0	No effect
0b1	Deactivate the trigger.

Accessing this field has the following behavior:

- When n >= UInt(CTIDEVID.NUMTRIG), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **WO/UNKNOWN**.

Accessing CTIINTACK

A debugger must read CTITRIGOUTSTATUS to confirm that the output trigger has been acknowledged before generating any event that must be ordered after the write to CTIINTACK, such as a write to CTIAPPPULSE to activate another trigger.

CTIINTACK can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x010	CTIINTACK

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **WI**.
- Otherwise, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIITCTRL, CTI Integration mode Control register

The CTIITCTRL characteristics are:

Purpose

Enables the CTI to switch from its default mode into integration mode, where test software can control directly the inputs and outputs of the PE, for integration testing or topology detection.

Configuration

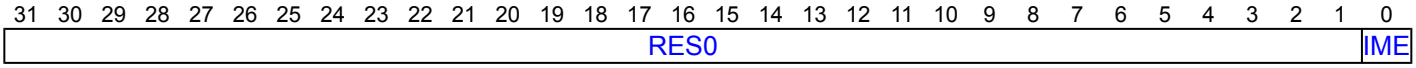
The power domain of CTIITCTRL is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

Attributes

CTIITCTRL is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

IME, bit [0]

Integration mode enable. When IME == 1, the device reverts to an integration mode to enable integration testing or topology detection.

IME	Meaning
0b0	Normal operation.
0b1	Integration mode enabled.

The integration mode behavior is IMPLEMENTATION DEFINED.

The following resets apply:

- If the register is implemented in the Core power domain:
  - On a Cold reset, this field resets to 0.
  - On an External debug reset, the value of this field is unchanged.
  - On a Warm reset, the value of this field is unchanged.
- If the register is implemented in the External debug power domain:
  - On a Cold reset, the value of this field is unchanged.
  - On an External debug reset, this field resets to 0.
  - On a Warm reset, the value of this field is unchanged.

Accessing CTIITCTRL

CTIITCTRL can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xF00	CTIITCTRL

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register are **IMPDEF**.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTILAR, CTI Lock Access Register

The CTILAR characteristics are:

Purpose

Allows or disallows access to the CTI registers through a memory-mapped interface.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Cross-Trigger Interface registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Cross-Trigger Interface registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

CTILAR is in the Debug power domain.

If FEAT_Debugv8p4 is implemented, the Software Lock is not implemented.

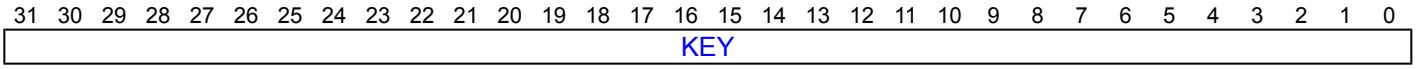
Software uses CTILAR to set or clear the lock, and CTILSR to check the current status of the lock.

Attributes

CTILAR is a 32-bit register.

Field descriptions

When CTI Software Lock is implemented:

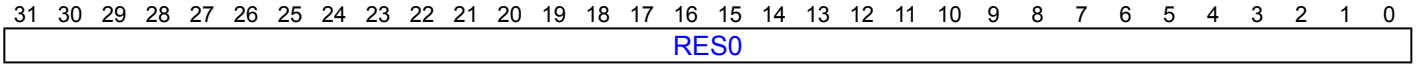


KEY, bits [31:0]

Lock Access control. Writing the key value `0xC5ACCE55` to this field unlocks the lock, enabling write accesses to this component's registers through a memory-mapped interface.

Writing any other value to this register locks the lock, disabling write accesses to this component's registers through a memory mapped interface.

Otherwise:



Otherwise

Bits [31:0]

Reserved, RES0.

Accessing CTILAR

CTILAR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance

CTI	0xFB0	CTILAR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTILSR, CTI Lock Status Register

The CTILSR characteristics are:

Purpose

Indicates the current status of the Software Lock for CTI registers.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Cross-Trigger Interface registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Cross-Trigger Interface registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

CTILSR is in the Debug power domain.

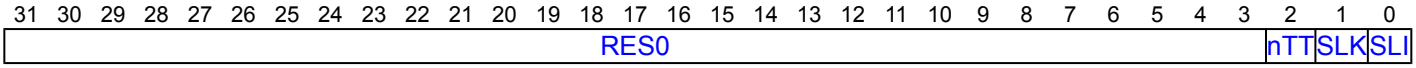
If FEAT_Debugv8p4 is implemented, the Software Lock is not implemented.

Software uses CTILAR to set or clear the lock, and CTILSR to check the current status of the lock.

Attributes

CTILSR is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

nTT, bit [2]

Not thirty-two bit access required. RAZ.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SLK, bit [1]

When CTI Software Lock is implemented:

Software Lock status for this component. For an access to LSR that is not a memory-mapped access, or when the Software Lock is not implemented, this field is RES0.

For memory-mapped accesses when the Software Lock is implemented, possible values of this field are:

SLK	Meaning
0b0	Lock clear. Writes are permitted to this component's registers.
0b1	Lock set. Writes to this component's registers are ignored, and reads have no side effects.

The reset behavior of this field is:

- On an External debug reset, this field resets to '1'.

Otherwise:

Reserved, RAZ.

SLI, bit [0]

Software Lock implemented. For an access to LSR that is not a memory-mapped access, this field is RAZ.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SLI	Meaning
0b0	Software Lock not implemented or not memory-mapped access.
0b1	Software Lock implemented and memory-mapped access.

Access to this field is **RO**.

Accessing CTILSR

CTILSR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
CTI	0xFB4	CTILSR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIOUTEN<n>, CTI Input Channel to Output Trigger Enable registers, n = 0 - 31

The CTIOUTEN<n> characteristics are:

Purpose

Defines which input channels generate output trigger n.

Configuration

CTIOUTEN<n> is in the Debug power domain.

If output trigger n is not implemented or not connected, CTIOUTEN<n> is RES0.

Attributes

CTIOUTEN<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	
OUTEN31	OUTEN30	OUTEN29	OUTEN28	OUTEN27	OUTEN26	OUTEN25	OUTEN24	OUTEN23	OUTEN22	OUTEN21	OUTEN20	OU

OUTEN<x>, bit [x], for x = 31 to 0

Input channel <x> to output trigger <n> enable.

OUTEN<x>	Meaning
0b0	An event on input channel <x> will not cause output trigger <n> to be asserted.
0b1	An event on input channel <x> will cause output trigger <n> to be asserted.

The reset behavior of this field is:

- On an External debug reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When x >= UInt(CTIDEVID.NUMCHAN), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RW**.

Accessing CTIOUTEN<n>

CTIOUTEN<n> can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x0A0 + (4 * n)	CTIOUTEN<n>

Accessible as follows:

- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIPIDR0, CTI Peripheral Identification Register 0

The CTIPIDR0 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

CTIPIDR0 is in the Debug power domain.

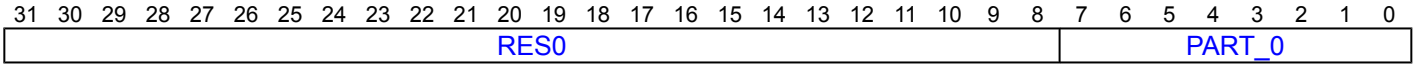
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTIPIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, least significant byte.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIPIDR0

CTIPIDR0 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFE0	CTIPIDR0

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIPIDR1, CTI Peripheral Identification Register 1

The CTIPIDR1 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

CTIPIDR1 is in the Debug power domain.

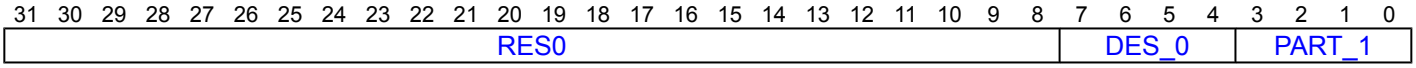
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTIPIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, least significant nibble of JEP106 ID code. For Arm Limited, this field is `0b1011`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, most significant nibble.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIPIDR1

CTIPIDR1 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	`0xFE4`	CTIPIDR1

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIPIDR2, CTI Peripheral Identification Register 2

The CTIPIDR2 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

CTIPIDR2 is in the Debug power domain.

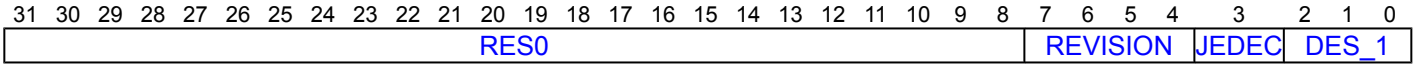
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTIPIDR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Part major revision. Parts can also use this field to extend Part number to 16-bits.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

Indicates a JEP106 identity code is used.

Reads as `0b1`.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, most significant bits of JEP106 ID code. For Arm Limited, this field is `0b011`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIPIDR2

CTIPIDR2 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFE8	CTIPIDR2

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIPIDR3, CTI Peripheral Identification Register 3

The CTIPIDR3 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

CTIPIDR3 is in the Debug power domain.

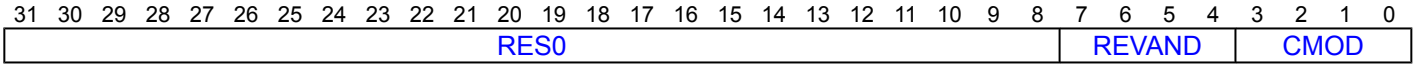
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTIPIDR3 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Part minor revision. Parts using CTIPIDR2.REVISION as an extension to the Part number must use this field as a major revision number.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer modified. Indicates someone other than the Designer has modified the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIPIDR3

CTIPIDR3 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0xFEC	CTIPIDR3

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIPIDR4, CTI Peripheral Identification Register 4

The CTIPIDR4 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

CTIPIDR4 is in the Debug power domain.

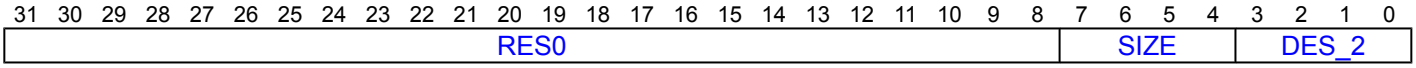
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTIPIDR4 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component. Log_2 of the number of 4KB pages from the start of the component to the end of the component ID registers.

Reads as `0b0000`.

Access to this field is **RO**.

DES_2, bits [3:0]

Designer, JEP106 continuation code, least significant nibble. For Arm Limited, this field is `0b0100`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIPIDR4

CTIPIDR4 can be accessed through the external debug interface:

Component	Offset	Instance
CTI	`0xFD0`	CTIPIDR4

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTITRIGINSTATUS, CTI Trigger In Status register

The CTITRIGINSTATUS characteristics are:

Purpose

Provides the status of the trigger inputs.

Configuration

CTITRIGINSTATUS is in the Debug power domain.

Attributes

CTITRIGINSTATUS is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
TRIN31	TRIN30	TRIN29	TRIN28	TRIN27	TRIN26	TRIN25	TRIN24	TRIN23	TRIN22	TRIN21	TRIN20	TRIN19	TRIN18	TRIN17	TRIN16	TR

TRIN<n>, bit [n], for n = 31 to 0

Trigger input <n> status.

TRIN<n>	Meaning
0b0	Input trigger n is inactive.
0b1	Input trigger n is active.

Not implemented and not-connected input triggers are always inactive.

It is IMPLEMENTATION DEFINED whether an input trigger that does not support multicycle events can be observed as active.

Accessing this field has the following behavior:

- When n >= UInt(CTIDEVID.NUMTRIG), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RO**.

Accessing CTITRIGINSTATUS

CTITRIGINSTATUS can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x130	CTITRIGINSTATUS

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTITRIGOUTSTATUS, CTI Trigger Out Status register

The CTITRIGOUTSTATUS characteristics are:

Purpose

Provides the raw status of the trigger outputs, after processing by any IMPLEMENTATION DEFINED trigger interface logic. For output triggers that are self-acknowledging, this is only meaningful if the CTI implements multicycle channel events.

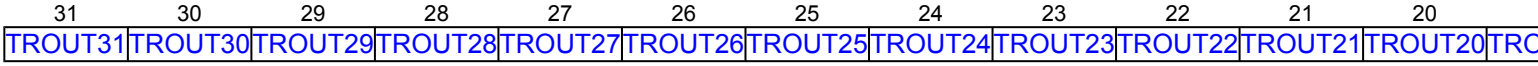
Configuration

CTITRIGOUTSTATUS is in the Debug power domain.

Attributes

CTITRIGOUTSTATUS is a 32-bit register.

Field descriptions



TROUT<n>, bit [n], for n = 31 to 0

Trigger output <n> status.

If n is less than CTIDEVID.NUMTRIG, and output trigger <n> is implemented and connected, and either the trigger is not self-acknowledging or the CTI implements multicycle channel events, then permitted values for TROUT<n> are:

TROUT<n>	Meaning
0b0	Output trigger n is inactive.
0b1	Output trigger n is active.

Otherwise when n is less than CTIDEVID.NUMTRIG, it is IMPLEMENTATION DEFINED whether TROUT<n> behaves as described here or is RAZ.

Accessing this field has the following behavior:

- When n >= UInt(CTIDEVID.NUMTRIG), access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - Output trigger <n> is not implemented or not connected or (Output trigger <n> is self-acknowledging and CTI does not implement multicycle channel events).
  - an implementation implements TROUT<n>.
- Otherwise, access to this field is **RO**.

Accessing CTITRIGOUTSTATUS

CTITRIGOUTSTATUS can be accessed through the external debug interface:

Component	Offset	Instance
CTI	0x134	CTITRIGOUTSTATUS

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGAUTHSTATUS_EL1, Debug Authentication Status Register

The DBGAUTHSTATUS_EL1 characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

Configuration

External register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to AArch64 System register DBGAUTHSTATUS_EL1[31:0].

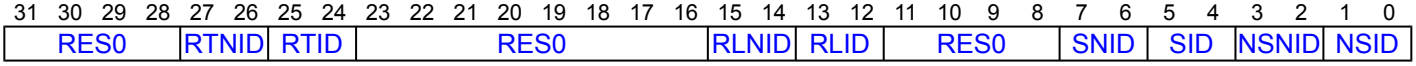
External register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGAUTHSTATUS[31:0].

When FEAT_DoPD is implemented, DBGAUTHSTATUS_EL1 is in the Core power domain. Otherwise, DBGAUTHSTATUS_EL1 is in the Debug power domain.

Attributes

DBGAUTHSTATUS_EL1 is a 32-bit register.

Field descriptions



Bits [31:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTID.

RTID, bits [25:24]

Root invasive debug.

RTID	Meaning
0b00	Not implemented.
0b10	Implemented and disabled. ExternalRootInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalRootInvasiveDebugEnabled() == TRUE.

All other values are reserved.

If FEAT_RME is not implemented, the only permitted value is 0b00.

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLID.

RLID, bits [13:12]

Realm invasive debug.

RLID	Meaning
0b00	Not implemented.
0b10	Implemented and disabled. ExternalRealmInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalRealmInvasiveDebugEnabled() == TRUE.

All other values are reserved.

If FEAT_RME is not implemented, the only permitted value is 0b00.

Bits [11:8]

Reserved, RES0.

SNID, bits [7:6]

When FEAT_Debugv8p4 is implemented:

Secure non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.SID.

Otherwise:

Secure non-invasive debug.

SNID	Meaning
0b00	Secure state is not implemented.
0b10	Implemented and disabled. ExternalSecureNoninvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalSecureNoninvasiveDebugEnabled() == TRUE.

All other values are reserved.

SID, bits [5:4]

Secure invasive debug.

SID	Meaning
0b00	Secure state is not implemented.
0b10	Implemented and disabled. ExternalSecureInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalSecureInvasiveDebugEnabled() == TRUE.

All other values are reserved.

NSNID, bits [3:2]

When FEAT_Debugv8p4 is implemented:

Non-secure non-invasive debug.

NSNID	Meaning
0b00	Non-secure state is not implemented.
0b11	Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE.

If the Effective value of SCR_EL3.NS is 1, or if EL3 is implemented and EL2 is not implemented, this field reads as 0b11.

All other values are reserved.

Otherwise:

Non-secure non-invasive debug.

NSNID	Meaning
0b00	Non-secure state is not implemented.
0b10	Implemented and disabled. ExternalNoninvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE.

All other values are reserved.

NSID, bits [1:0]

Non-secure invasive debug.

NSID	Meaning
0b00	Non-secure state is not implemented.
0b10	Implemented and disabled. ExternalInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalInvasiveDebugEnabled() == TRUE.

All other values are reserved.

Accessing DBGAUTHSTATUS_EL1

DBGAUTHSTATUS_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFB8	DBGAUTHSTATUS_EL1

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGBCR<n>_EL1, Debug Breakpoint Control Registers, n = 0 - 63

The DBGBCR<n>_EL1 characteristics are:

Purpose

Holds control information for a breakpoint. Forms breakpoint n together with value register DBGBVR<n>_EL1.

Configuration

External register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to AArch64 System register DBGBCR<n>_EL1[31:0].

External register DBGBCR<n>_EL1 bits [63:32] are architecturally mapped to AArch64 System register DBGBCR<n>_EL1[63:32] when FEAT_Debugv8p9 is implemented.

External register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGBCR<n>[31:0].

DBGBCR<n>_EL1 is in the Core power domain.

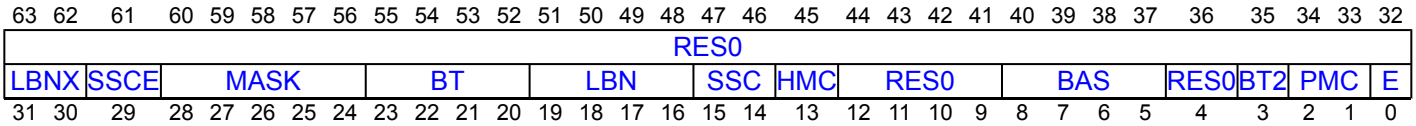
If breakpoint n is not implemented then accesses to this register are:

- RES0 when IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalDebugAccess().
- A CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR otherwise.

Attributes

DBGBCR<n>_EL1 is a 64-bit register.

Field descriptions



When the E field is zero, all the other fields in the register are ignored.

Bits [63:32]

Reserved, RES0.

LBNX, bits [31:30]

When FEAT_Debugv8p9 is implemented:

Linked Breakpoint Number.

For Linked address matching breakpoints, with DBGBCR<n>_EL1.LBN, specifies the index of the breakpoint linked to.

For all other breakpoint types, this field is ignored and reads of the register return an UNKNOWN value.

This field extends DBGBCR<n>_EL1.LBN to support up to 64 implemented breakpoints.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSCE, bit [29]
When FEAT_RME is implemented:

Security State Control Extended.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MASK, bits [28:24]
When FEAT_BWE is implemented:

Address Mask. Only address ranges up to 2GB can be watched using a single mask.

MASK	Meaning
0b00000	No mask.
0b00011..0b11111	Number of address bits masked.

All other values are reserved.

Indicates the number of masked address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b11111 masking 31 address bits (0x7FFFFFFF mask for address).

If any of the following apply then the behavior of the breakpoint is CONSTRAINED UNPREDICTABLE:

- DBGBCR<n>_EL1.MASK is a reserved value.
- DBGBCR<n>_EL1.MASK is zero, DBGBCR<n>_EL1.{BT2, BT} is {0, 0b010x} (address mismatch breakpoint without linking enabled), and AArch32 is not supported at EL1.
- DBGBCR<n>_EL1.MASK is a valid nonzero value and any of the following apply:
  - DBGBCR<n>_EL1.BAS is not 0b1111, and AArch32 is supported at EL0.
  - DBGBVR<n>_EL1[(MASK-1):0] is nonzero.
  - DBGBCR<n>_EL1.{BT2, BT} is {0, 0b0x1x} or {0, 0b1xxx} (Context matching breakpoint).

When any of these conditions apply, the breakpoint behaves as if one of the following:

- DBGBCR<n>_EL1.MASK has been programmed with a defined value, which might be 0b00000 (no mask), other than for a direct read of DBGBCR<n>_EL1.
- The breakpoint is disabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bits [23:20]

Breakpoint Type.

With DBGBCR<n>_EL1.BT2 when implemented, specifies breakpoint type.

BT	Meaning	Applies when
0b0000	Unlinked instruction address match. DBGBVR<n>_EL1 is the address of an instruction.	
0b0001	Linked instruction address match. As 0b0000, but linked to a breakpoint that has linking enabled.	
0b0010	Unlinked Context ID match. If the Effective value of HCR_EL2.E2H is 1 and either the PE is executing at EL0 with HCR_EL2.TGE set to 1 or the PE is executing at EL2, then DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL2. Otherwise, DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL1.	When breakpoint n is context-aware
0b0011	As 0b0010, with linking enabled.	When breakpoint n is context-aware
0b0100	Unlinked instruction address mismatch. DBGBVR<n>_EL1 is the address of an instruction.	When FEAT_BWE is implemented or EL1 is using AArch32
0b0101	Linked instruction address mismatch. As 0b0100, but linked to a breakpoint that has linking enabled.	When FEAT_BWE is implemented or EL1 is using AArch32
0b0110	Unlinked CONTEXTIDR_EL1 match. DBGBVR<n>_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1.	When EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b0111	As 0b0110, with linking enabled.	When EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1000	Unlinked VMID match. DBGBVR<n>_EL1.VMID is a VMID compared against VTTBR_EL2.VMID.	When EL2 is implemented and breakpoint n is context-aware
0b1001	As 0b1000, with linking enabled.	When EL2 is implemented and breakpoint n is context-aware
0b1010	Unlinked VMID and Context ID match. DBGBVR<n>_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1, and DBGBVR<n>_EL1.VMID is a VMID compared against VTTBR_EL2.VMID.	When EL2 is implemented and breakpoint n is context-aware
0b1011	As 0b1010, with linking enabled.	When EL2 is implemented and breakpoint n is context-aware
0b1100	Unlinked CONTEXTIDR_EL2 match. DBGBVR<n>_EL1.ContextID2 is a Context ID compared against CONTEXTIDR_EL2.	When EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1101	As 0b1100, with linking enabled.	When EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1110	Unlinked Full Context ID match. DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL1, and	When EL2 is implemented, FEAT_Debugv8p1 is implemented,

	DBGBVR<n>_EL1.ContextID2 is compared against CONTEXTIDR_EL2.	and breakpoint n is context-aware
0b1111	As 0b1110, with linking enabled.	When EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked Breakpoint Number.

For Linked address matching breakpoints, with DBGBCR<n>_EL1.LBNX when implemented, specifies the index of the breakpoint linked to.

For all other breakpoint types, this field is ignored and reads of the register return an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the HMC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information, including the effect of programming the fields to a reserved set of values, see 'Reserved DBGBCR<n>_EL1.{SSC, HMC, PMC} values'.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see DBGBCR<n>_EL1.SSC description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [12:9]

Reserved, RES0.

BAS, bits [8:5]

When FEAT_AA32 is implemented:

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state.

The permitted values depend on the breakpoint type.

For Address match breakpoints in either AArch32 or AArch64 state, the permitted values are:

BAS	Match instruction at	Constraint for debuggers
0b0011	DBGBVR<n>_EL1	Use for T32 instructions.
0b1100	DBGBVR<n>_EL1 + 2	Use for T32 instructions.
0b1111	DBGBVR<n>_EL1	Use for A64 and A32 instructions.

All other values are reserved.

For more information, see 'Using the BAS field in Address Match breakpoints'.

For Address mismatch breakpoints in an AArch32 stage 1 translation regime, the permitted values are:

BAS	Match instruction at	Constraint for debuggers
0b0000	-	Use for a match anywhere breakpoint.
0b0011	DBGBVR<n>_EL1	Use for stepping T32 instructions.
0b1100	DBGBVR<n>_EL1 + 2	Use for stepping T32 instructions.
0b1111	DBGBVR<n>_EL1	Use for stepping A64 and A32 instructions.

All other values are reserved.

For more information, see 'Using the BAS field in Address Match breakpoints'.

For Context matching breakpoints, this field is RES1 and ignored.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [4]

Reserved, RES0.

BT2, bit [3]

When FEAT_ABLE is implemented and breakpoint n supports address breakpoint linking:

Breakpoint Type 2. With DBGBCR<n>_EL1.BT, specifies breakpoint type.

BT2	Meaning
0b0	As DBGBCR<n>_EL1.BT.
0b1	As DBGBCR<n>_EL1.BT, but with linking enabled. This value is only defined for the following DBGBCR<n>_EL1.BT values: 0b0000, 0b0001, 0b0100, and 0b0101. All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMC, bits [2:1]

Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and HMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the DBGBCR<n>_EL1.SSC description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable breakpoint n.

E	Meaning
0b0	Breakpoint n disabled.
0b1	Breakpoint n enabled.

This field is ignored by the PE and treated as zero when all of the following are true:

- Any of the following are true:
  - `HaltOnBreakpointOrWatchpoint()` is FALSE and the Effective value of MDSCR_EL1.EMBWE is 0.
  - `HaltOnBreakpointOrWatchpoint()` is TRUE and the Effective value of EDSCR2.EHBWE is 0.
- FEAT_Debugv8p9 is implemented.
- n >= 16.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBCR<n>_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

When FEAT_Debugv8p9 is not implemented, this register is 32-bits wide and offset `0x40C` + (16 * n) is reserved.

DBGBCR<n>_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0x408` + (16 * n)	DBGBCR<n>_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalDebugAccess(addrdesc), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGBVR<n>_EL1, Debug Breakpoint Value Registers, n = 0 - 63

The DBGBVR<n>_EL1 characteristics are:

Purpose

Holds a virtual address, or a VMID and/or a context ID, for use in breakpoint matching. Forms breakpoint n together with control register DBGBCR<n>_EL1.

Configuration

External register DBGBVR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register DBGBVR<n>_EL1[63:0].

External register DBGBVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGBVR<n>[31:0].

External register DBGBVR<n>_EL1 bits [63:32] are architecturally mapped to AArch32 System register DBGBXVR<n>[31:0].

DBGBVR<n>_EL1 is in the Core power domain.

How this register is interpreted depends on the value of DBGBCR<n>_EL1.BT.

- When DBGBCR<n>_EL1.BT is `0b0x0x`, this register holds a virtual address.
- When DBGBCR<n>_EL1.BT is `0b001x`, `0b011x`, or `0b110x`, this register holds a Context ID.
- When DBGBCR<n>_EL1.BT is `0b100x`, this register holds a VMID.
- When DBGBCR<n>_EL1.BT is `0b101x`, this register holds a VMID and a Context ID.
- When DBGBCR<n>_EL1.BT is `0b111x`, this register holds two Context ID values.

For other values of DBGBCR<n>_EL1.BT, this register is RES0.

If breakpoint n is not implemented then accesses to this register are:

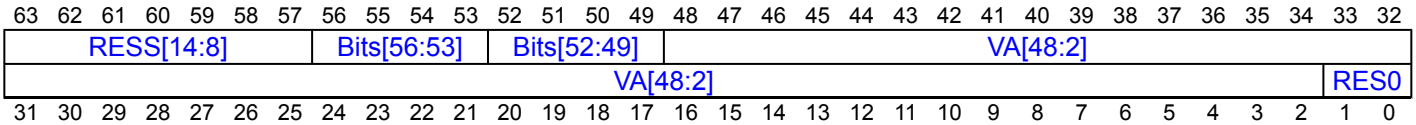
- RES0 when IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalDebugAccess().
- A CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR otherwise.

Attributes

DBGBVR<n>_EL1 is a 64-bit register.

Field descriptions

When DBGBCR<n>_EL1.BT IN {0b0x0x}:



RESS[14:8], bits [63:57]

Reserved, Sign extended. Software must treat this field as RES0 if the most significant bit of VA is 0 or RES0, and as RES1 if the most significant bit of VA is 1.

Hardware always ignores the value of these bits and it is IMPLEMENTATION DEFINED whether:

- The bits are hardwired to a copy of the most significant bit of VA, meaning writes to these bits are ignored, and reads to the bits always return the hardwired value.
- The value in those bits can be written, and reads will return the last value written. The value held in those bits is ignored by hardware.

Bits[56:53]

When FEAT_LVA3 is implemented:

VA[56:53], bits [3:0] of bits [56:53]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[7:4], bits [3:0] of bits [56:53]

Extension to RESS[14:8]. For more information, see RESS[14:8].

Bits[52:49]

When FEAT_LVA is implemented:

VA[52:49], bits [3:0] of bits [52:49]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[3:0], bits [3:0] of bits [52:49]

Extension to RESS[14:8]. For more information, see RESS[14:8].

VA[48:2], bits [48:2]

If the address is being matched in an AArch64 stage 1 translation regime:

- This field contains bits[48:2] of the address for comparison.
- When FEAT_LVA3 is implemented, (VA[56:53]:VA[52:49]) forms the upper part of the address value. If FEAT_LVA3 is not implemented, bits VA[56:53] are part of the RESS field.
- When FEAT_LVA is implemented, VA[52:49] forms the upper part of the address value. If FEAT_LVA is not implemented, bits [52:49] are part of the RESS field.

If the address is being matched in an AArch32 stage 1 translation regime, the first 20 bits of this field are RES0, and the rest of the field contains bits[31:2] of the address for comparison.

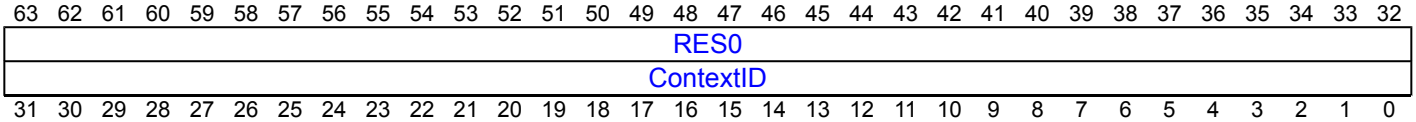
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT IN {0b001x}:



Bits [63:32]

Reserved, RES0.

ContextID, bits [31:0]

Context ID value for comparison.

The value is compared against CONTEXTIDR_EL2 when the Effective value of HCR_EL2.E2H is 1, and either:

- The PE is executing at EL2.
- HCR_EL2.TGE is 1, the PE is executing at EL0, and EL2 is enabled in the current Security state.

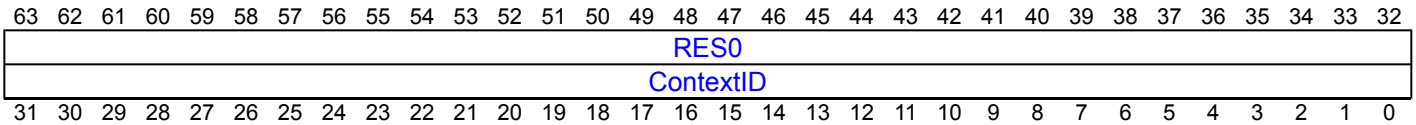
Otherwise, the value is compared against the following:

- CONTEXTIDR when the PE is executing at AArch32.
- CONTEXTIDR_EL1 when the PE is executing at AArch64.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT IN {0b011x}, EL2 is implemented, and FEAT_Debugv8p1 is implemented:



Bits [63:32]

Reserved, RES0.

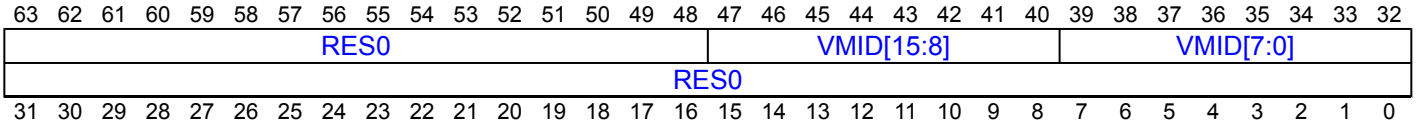
ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT IN {0b100x} and EL2 is implemented:



Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VMID16 is implemented and VTCR_EL2.VS == 1:

Extension to VMID[7:0]. For more information, see DBGBVR<n>_EL1.VMID[7:0].

If EL2 is using AArch32, this field is RES0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

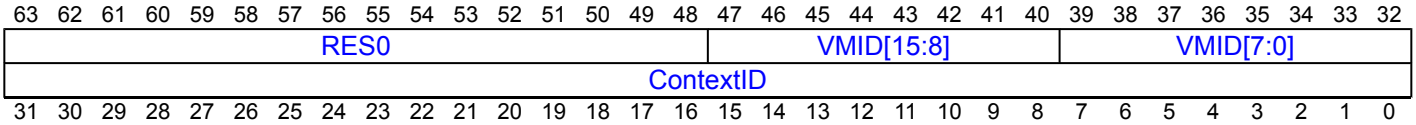
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT IN {0b101x} and EL2 is implemented:



Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VMID16 is implemented and VTCR_EL2.VS == 1:

Extension to VMID[7:0]. For more information, see DBGBVR<n>_EL1.VMID[7:0].

If EL2 is using AArch32, or if the implementation has an 8-bit VMID, this field is RES0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

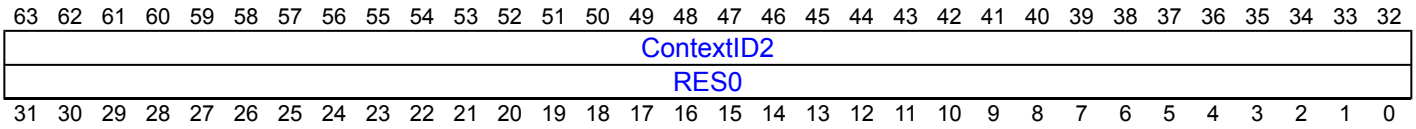
ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT IN {0b110x}, EL2 is implemented, and FEAT_Debugv8p1 is implemented:



ContextID2, bits [63:32]

Context ID value for comparison against CONTEXTIDR_EL2.

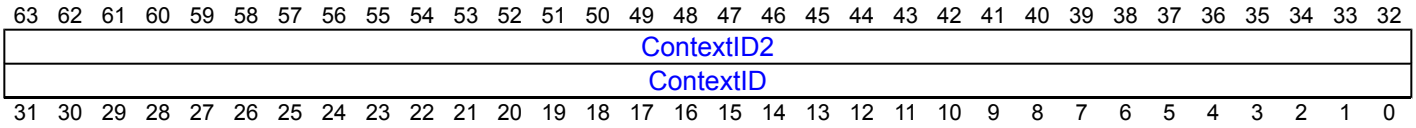
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT IN {0b111x}, EL2 is implemented, and FEAT_Debugv8p1 is implemented:



ContextID2, bits [63:32]

Context ID value for comparison against CONTEXTIDR_EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBVR<n>_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

DBGBVR<n>_EL1 can be accessed through the external debug interface:

Component	Offset	Instance	Range
Debug	0x400 + (16 * n)	DBGBVR<n>_EL1	63:0

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalDebugAccess(addrdesc), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMCLR_EL1, Debug CLAIM Tag Clear Register

The DBGCLAIMCLR_EL1 characteristics are:

Purpose

Used by software to read the values of the CLAIM tag bits, and to clear CLAIM tag bits to 0.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMSET_EL1 register.

Configuration

External register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMCLR_EL1[31:0].

External register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMSET_EL1[31:0].

External register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMCLR[31:0].

External register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMSET[31:0].

External register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMSET_EL1[31:0].

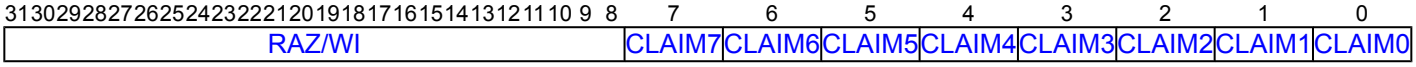
DBGCLAIMCLR_EL1 is in the Core power domain.

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMCLR_EL1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RAZ/WI.

CLAIM<m>, bit [m], for m = 7 to 0

Claim Tag Clear. Indicates the current status of Claim Tag bit <m>, and is used to clear Claim Tag bit <m> to 0.

CLAIM<m>	Meaning
0b0	On a read: Claim Tag bit <m> is not set. On a write: Ignored.
0b1	On a read: Claim Tag bit <m> is set. On a write: Clear Claim tag bit <m> to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Access to this field is **W1C**.

Accessing DBGCLAIMCLR_EL1

DBGCLAIMCLR_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFA4	DBGCLAIMCLR_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMSET_EL1, Debug CLAIM Tag Set Register

The DBGCLAIMSET_EL1 characteristics are:

Purpose

Used by software to set the CLAIM tag bits to 1.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMCLR_EL1 register.

Configuration

External register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMSET_EL1[31:0].

External register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMCLR_EL1[31:0].

External register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMSET[31:0].

External register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMCLR[31:0].

External register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMCLR_EL1[31:0].

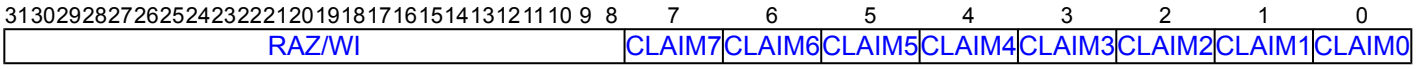
DBGCLAIMSET_EL1 is in the Core power domain.

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMSET_EL1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RAZ/WI.

CLAIM<m>, bit [m], for m = 7 to 0

Claim Tag Set. Used to set Claim Tag bit <m> to 1.

CLAIM<m>	Meaning
0b0	On a write: Ignored.
0b1	On a write: Set Claim Tag bit <m> to 1.

Access to this field is **RAO/W1S**.

Accessing DBGCLAIMSET_EL1

DBGCLAIMSET_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFA0	DBGCLAIMSET_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRRX_EL0, Debug Data Transfer Register, Receive

The DBGDTRRX_EL0 characteristics are:

Purpose

Transfers data from an external debugger to the PE. For example, it is used by a debugger transferring commands and data to a debug target. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communications Channel.

Configuration

External register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to AArch64 System register DBGDTRRX_EL0[31:0].

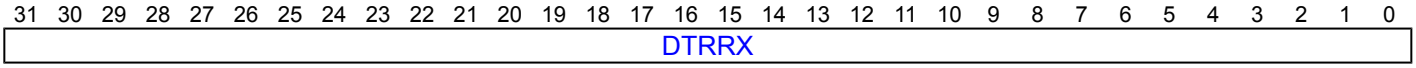
External register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRRXint[31:0].

DBGDTRRX_EL0 is in the Core power domain.

Attributes

DBGDTRRX_EL0 is a 32-bit register.

Field descriptions



DTRRX, bits [31:0]

Update DTRRX.

Writes to this register:

- If RXfull is 0, update the value in DTRRX and set RXfull to 1.
- If RXfull is 1, the written value is ignored and RXO is set to 1.

Reads of this register return the last value written to DTRRX and do not change RXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRRX_EL0

If EDSCR.ITE == 0 when the PE exits Debug state on receiving a Restart request trigger event, the behavior of any operation issued by a DTR access in memory access mode that has not completed execution is CONSTRAINED UNPREDICTABLE, and must do one of the following:

- It must complete execution in Debug state before the PE executes the restart sequence.
- It must complete execution in Non-debug state before the PE executes the restart sequence.
- It must be abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an UNKNOWN state.

DBGDTRRX_EL0 can be accessed through the external debug interface:

Component	Offset	Instance

Debug	0x080	DBGDTRRX_EL0

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRTX_EL0, Debug Data Transfer Register, Transmit

The DBGDTRTX_EL0 characteristics are:

Purpose

Transfers data from the PE to an external debugger. For example, it is used by a debug target to transfer data to the debugger. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communication Channel.

Configuration

External register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to AArch64 System register DBGDTRTX_EL0[31:0].

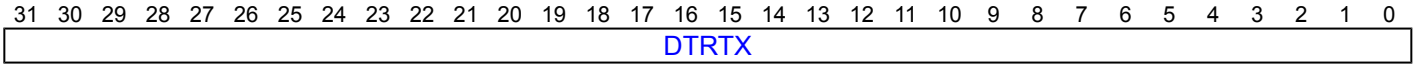
External register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXint[31:0].

DBGDTRTX_EL0 is in the Core power domain.

Attributes

DBGDTRTX_EL0 is a 32-bit register.

Field descriptions



DTRTX, bits [31:0]

Return DTRTX.

Reads of this register:

- If TXfull is 1, return the value in DTRTX and clear TXfull to 0.
- If TXfull is 0, return an UNKNOWN value and set TXU to 1.

Writes of this register update the value in DTRTX and do not change TXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRTX_EL0

If EDSCR.ITE == 0 when the PE exits Debug state on receiving a Restart request trigger event, the behavior of any operation issued by a DTR access in memory access mode that has not completed execution is CONSTRAINED UNPREDICTABLE, and must do one of the following:

- It must complete execution in Debug state before the PE executes the restart sequence.
- It must complete execution in Non-debug state before the PE executes the restart sequence.
- It must be abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an UNKNOWN state.

DBGDTRTX_EL0 can be accessed through the external debug interface:

Component	Offset	Instance

Debug	0x08C	DBGDTRTX_EL0

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalDebugAccess(addrdesc), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWCR<n>_EL1, Debug Watchpoint Control Registers, n = 0 - 63

The DBGWCR<n>_EL1 characteristics are:

Purpose

Holds control information for a watchpoint. Forms watchpoint n together with value register DBGWVR<n>_EL1.

Configuration

External register DBGWCR<n>_EL1 bits [31:0] are architecturally mapped to AArch64 System register DBGWCR<n>_EL1[31:0].

External register DBGWCR<n>_EL1 bits [63:32] are architecturally mapped to AArch64 System register DBGWCR<n>_EL1[63:32] when FEAT_Debugv8p9 is implemented.

External register DBGWCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGWCR<n>[31:0].

DBGWCR<n>_EL1 is in the Core power domain.

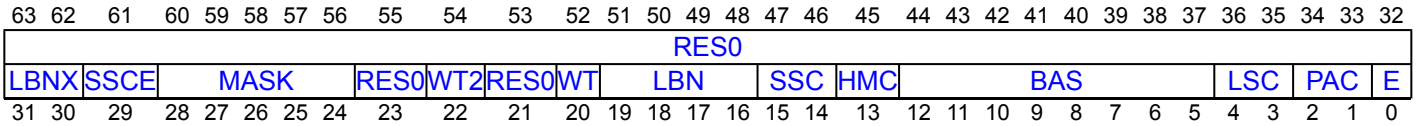
If watchpoint n is not implemented then accesses to this register are:

- When IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalDebugAccess(), RES0.
- Otherwise, a CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR.

Attributes

DBGWCR<n>_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

LBNX, bits [31:30]

When FEAT_Debugv8p9 is implemented:

Linked Breakpoint Number.

For Linked data address watchpoints, with DBGWCR<n>_EL1.LBN, specifies the index of the breakpoint linked to.

For all other watchpoint types, this field is ignored and reads of the register return an UNKNOWN value.

This field extends DBGWCR<n>_EL1.LBN to support up to 64 implemented breakpoints.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSCE, bit [29]

When FEAT_RME is implemented:

Security State Control Extended.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MASK, bits [28:24]

Address Mask. Only address ranges up to 2GB can be watched using a single mask.

MASK	Meaning
0b00000	No mask.
0b00011..0b11111	Number of address bits masked.

All other values are reserved.

Indicates the number of masked address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b11111 masking 31 address bits (0x7FFFFFFF mask for address).

If programmed with a reserved value, the watchpoint behaves as if one of the following:

- DBGWCR<n>_EL1.MASK has been programmed with a defined value, which might be 0b00000 (no mask), other than for a direct read of DBGWCR<n>_EL1.
- The watchpoint is disabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [23]

Reserved, RES0.

WT2, bit [22]

When FEAT_BWE2 is implemented:

Watchpoint Type 2. With DBGWCR<n>_EL1.WT, specifies watchpoint type.

WT2	Meaning
0b0	Watchpoint n is an address match watchpoint.
0b1	Watchpoint n is an address mismatch watchpoint.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [21]

Reserved, RES0.

WT, bit [20]

Watchpoint type. Possible values are:

WT	Meaning
0b0	Unlinked data address match.
0b1	Linked data address match.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked Breakpoint Number.

For Linked data address watchpoints, with DBGWCR<n>_EL1.LBNX when implemented, specifies the index of the breakpoint linked to.

For all other watchpoint types, this field is ignored and reads of the register return an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the HMC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

BAS, bits [12:5]

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by DBGWVR<n>_EL1 is being watched.

BAS	Description
xxxxxxx1	Match byte at DBGWVR<n>_EL1
xxxxxx1x	Match byte at DBGWVR<n>_EL1 + 1
xxxxx1xx	Match byte at DBGWVR<n>_EL1 + 2
xxxx1xxx	Match byte at DBGWVR<n>_EL1 + 3

In cases where DBGWVR<n>_EL1 addresses a double-word:

BAS	Description, if DBGWVR<n>_EL1[2] == 0
xxx1xxxx	Match byte at DBGWVR<n>_EL1 + 4
xx1xxxxx	Match byte at DBGWVR<n>_EL1 + 5
x1xxxxxx	Match byte at DBGWVR<n>_EL1 + 6
1xxxxxxx	Match byte at DBGWVR<n>_EL1 + 7

If DBGWVR<n>_EL1[2] == 1, only BAS[3:0] is used. Arm deprecates setting DBGWVR<n>_EL1[2] == 1.

The valid values for BAS are nonzero binary number all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<n>.BAS values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LSC, bits [4:3]

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

LSC	Meaning
0b01	Match instructions that load from a watchpointed address.
0b10	Match instructions that store to a watchpointed address.
0b11	Match instructions that load from or store to a watchpointed address.

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PAC, bits [2:1]

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and HMC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable watchpoint n.

E	Meaning
0b0	Watchpoint n disabled.
0b1	Watchpoint n enabled.

This field is ignored by the PE and treated as zero when all of the following are true:

- Any of the following are true:
  - `HaltOnBreakpointOrWatchpoint()` is FALSE and the Effective value of MDSCR_EL1.EMBWE is 0.
  - `HaltOnBreakpointOrWatchpoint()` is TRUE and the Effective value of EDSCR2.EHBWE is 0.
- FEAT_Debugv8p9 is implemented.
- n >= 16.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGWCR<n>_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

When FEAT_Debugv8p9 is not implemented, this register is 32-bits wide and offset `0x80C` + (16 * n) is reserved.

DBGWCR<n>_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0x808` + (16 * n)	DBGWCR<n>_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalDebugAccess(addrdesc), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWVR<n>_EL1, Debug Watchpoint Value Registers, n = 0 - 63

The DBGWVR<n>_EL1 characteristics are:

Purpose

Holds a data address value for use in watchpoint matching. Forms watchpoint n together with control register DBGWCR<n>_EL1.

Configuration

External register DBGWVR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register DBGWVR<n>_EL1[63:0].

External register DBGWVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGWVR<n>[31:0].

DBGWVR<n>_EL1 is in the Core power domain.

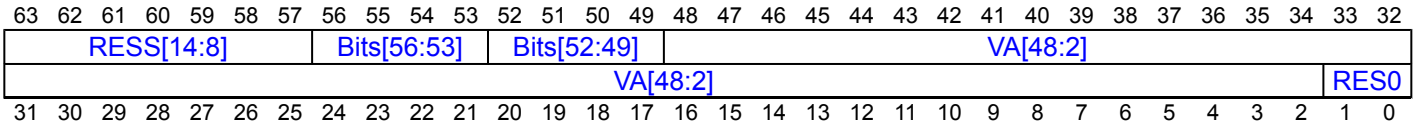
If watchpoint n is not implemented then accesses to this register are:

- When IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalDebugAccess(), RES0.
- Otherwise, a CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR.

Attributes

DBGWVR<n>_EL1 is a 64-bit register.

Field descriptions



RESS[14:8], bits [63:57]

Reserved, Sign extended. Hardware and software must treat this field as RES0 if the most significant bit of VA is 0 or RES0, and as RES1 if the most significant bit of VA is 1.

Hardware always ignores the value of these bits and it is IMPLEMENTATION DEFINED whether:

- The bits are hardwired to a copy of the most significant bit of VA, meaning writes to these bits are ignored, and reads to the bits always return the hardwired value.
- The value in those bits can be written, and reads will return the last value written. The value held in those bits is ignored by hardware.

Bits[56:53]

When FEAT_LVA3 is implemented:

VA[56:53], bits [3:0] of bits [56:53]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[7:4], bits [3:0] of bits [56:53]

Extension to RESS[14:8]. For more information, see RESS[14:8].

Bits[52:49]

When FEAT_LVA is implemented:

VA[52:49], bits [3:0] of bits [52:49]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[3:0], bits [3:0] of bits [52:49]

Extension to RESS[14:8]. For more information, see RESS[14:8].

VA[48:2], bits [48:2]

Bits[48:2] of the address value for comparison.

When FEAT_LVA3 is implemented, (VA[56:53]:VA[52:49]) forms the upper part of the address value. If FEAT_LVA3 is not implemented, bits VA[56:53] are part of the RESS field.

When FEAT_LVA is implemented, VA[52:49] forms the upper part of the address value. If FEAT_LVA is not implemented, bits [52:49] are part of the RESS field.

Arm deprecates setting DBGWVR<n>_EL1[2] == 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

Accessing DBGWVR<n>_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

DBGWVR<n>_EL1 can be accessed through the external debug interface:

Component	Offset	Instance	Range
Debug	0x800 + (16 * n)	DBGWVR<n>_EL1	63:0

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalDebugAccess(addrdesc), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDAA32PFR, External Debug Auxiliary Processor Feature Register

The EDAA32PFR characteristics are:

Purpose

Provides information about implemented PE features.

Note

The register mnemonic, EDAA32PFR, is derived from previous definitions of this register that defined this register only when AArch64 was not supported.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

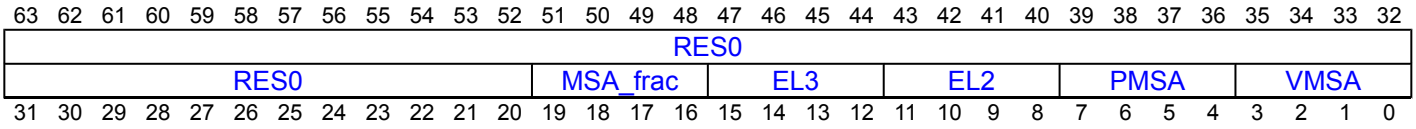
Configuration

There are no configuration notes.

Attributes

EDAA32PFR is a 64-bit register.

Field descriptions



Bits [63:20]

Reserved, RES0.

MSA_frac, bits [19:16]

When EDAA32PFR.PMSA == 0b0000 and EDAA32PFR.VMSA == 0b1111:

Memory System Architecture fractional field. This holds the information on additional Memory System Architectures supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MSA_frac	Meaning
0b0001	PMSAv8-64 supported in all translation regimes. VMSAv8-64 not supported.
0b0010	PMSAv8-64 supported in all translation regimes. In addition to PMSAv8-64, stage 1 EL1&0 translation regime also supports VMSAv8-64.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

EL3, bits [15:12]

When EDPFR.EL3 == 0b0000:

AArch32 EL3 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL3	Meaning
0b0000	EL3 is not implemented or can be executed in AArch64 state.
0b0001	EL3 can be executed in AArch32 state only.

All other values are reserved.

Note

EDPFR.{EL1, EL0} indicate whether EL1 and EL0 can only be executed in AArch32 state.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

EL2, bits [11:8]

When EDPFR.EL2 == 0b0000:

AArch32 EL2 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL2	Meaning
0b0000	EL2 is not implemented or can be executed in AArch64 state.
0b0001	EL2 can be executed in AArch32 state only.

All other values are reserved.

Note

EDPFR.{EL1, EL0} indicate whether EL1 and EL0 can only be executed in AArch32 state.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

PMSA, bits [7:4]

Indicates support for a 32-bit PMSA.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMSA	Meaning
0b0000	PMSA-32 not supported.
0b0100	PMSAv8-32 supported.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

VMSA, bits [3:0]

When EDAA32PFR.PMSA != 0b0000:

Indicates support for a VMSA in addition to a 32-bit PMSA.

VMSA	Meaning
0b0000	VMSA not supported.

All other values are reserved.

Access to this field is **RO**.

When EDAA32PFR.PMSA == 0b0000:

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMSA	Meaning
0b0000	VMSAv8-64 supported.
0b1111	Memory system architecture described by EDAA32PFR.MSA_frac.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

Accessing EDAA32PFR

EDAA32PFR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xD60	EDAA32PFR

Accessible as follows:

- When IsCorePowered() and !DoubleLockStatus(), accesses to this register are **RO**.
- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **IMPDEF**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDACR, External Debug Auxiliary Control Register

The EDACR characteristics are:

Purpose

Allows implementations to support IMPLEMENTATION DEFINED controls.

Configuration

When FEAT_DoPD is implemented, EDACR is in the Core power domain. Otherwise, it is IMPLEMENTATION DEFINED whether EDACR is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is OPTIONAL.

If this register is implemented, EDDEVID.AuxRegs == `0b0001`.

If FEAT_DoPD is implemented, any mechanism to preserve control bits in EDACR over power down is optional and IMPLEMENTATION DEFINED.

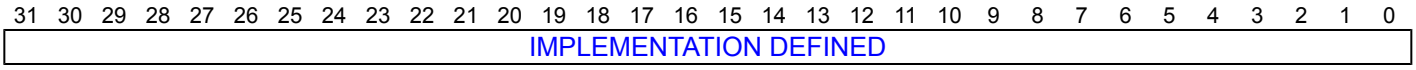
If FEAT_DoPD is not implemented and EDACR contains any control bits that must be preserved over power down, then these bits must be accessible by the external debug interface when the OS Lock is locked, OSLSR_EL1.OSLK == 1, and when the Core is powered off.

Changing this register from its reset value causes IMPLEMENTATION DEFINED behavior, including possible deviation from the architecturally-defined behavior.

Attributes

EDACR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The following resets apply:

- If the register is implemented in the Core power domain:
  - On a Cold reset, this field resets to an architecturally UNKNOWN value.
  - On an External debug reset, the value of this field is unchanged.
  - On a Warm reset, the value of this field is unchanged.
- If the register is implemented in the External debug power domain:
  - On a Cold reset, the value of this field is unchanged.
  - On an External debug reset, this field resets to an architecturally UNKNOWN value.
  - On a Warm reset, the value of this field is unchanged.

Accessing EDACR

EDACR can be accessed through the external debug interface:

Component	Offset	Instance

Debug	0x094	EDACR

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register are **IMPDEF**.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDCIDR0, External Debug Component Identification Register 0

The EDCIDR0 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Component Identification scheme'.

Configuration

When FEAT_DoPD is implemented, EDCIDR0 is in the Core power domain. Otherwise, EDCIDR0 is in the Debug power domain.

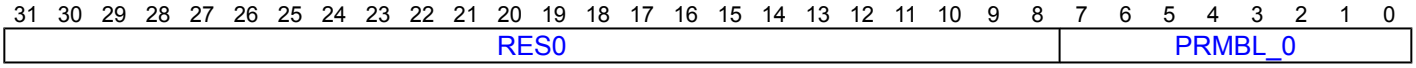
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

EDCIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Preamble.

Reads as `0x0D`.

Access to this field is **RO**.

Accessing EDCIDR0

EDCIDR0 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0xFF0`	EDCIDR0

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDCIDR1, External Debug Component Identification Register 1

The EDCIDR1 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Component Identification scheme'.

Configuration

When FEAT_DoPD is implemented, EDCIDR1 is in the Core power domain. Otherwise, EDCIDR1 is in the Debug power domain.

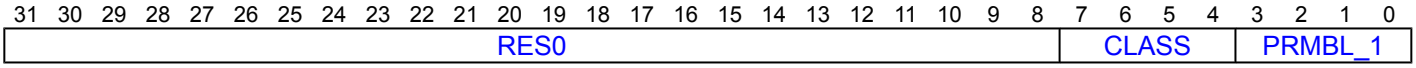
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

EDCIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

CLASS	Meaning
0b1001	CoreSight component.

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

Access to this field is **RO**.

PRMBL_1, bits [3:0]

Preamble.

Reads as 0b0000.

Access to this field is **RO**.

Accessing EDCIDR1

EDCIDR1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFF4	EDCIDR1

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDCIDR2, External Debug Component Identification Register 2

The EDCIDR2 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Component Identification scheme'.

Configuration

When FEAT_DoPD is implemented, EDCIDR2 is in the Core power domain. Otherwise, EDCIDR2 is in the Debug power domain.

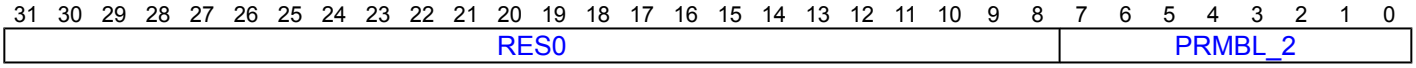
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

EDCIDR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Preamble.

Reads as `0x05`.

Access to this field is **RO**.

Accessing EDCIDR2

EDCIDR2 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0xFF8`	EDCIDR2

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDCIDR3, External Debug Component Identification Register 3

The EDCIDR3 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Component Identification scheme'.

Configuration

When FEAT_DoPD is implemented, EDCIDR3 is in the Core power domain. Otherwise, EDCIDR3 is in the Debug power domain.

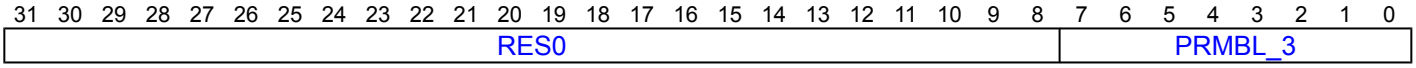
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

EDCIDR3 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Preamble.

Reads as `0xB1`.

Access to this field is **RO**.

Accessing EDCIDR3

EDCIDR3 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0xFFC`	EDCIDR3

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDCIDSR, External Debug Context ID Sample Register

The EDCIDSR characteristics are:

Purpose

Contains the sampled value of the Context ID, captured on reading EDPCSR[31:0].

Configuration

EDCIDSR is in the Core power domain.

This register is present only when FEAT_PCSRv8 is implemented and FEAT_PCSRv8p2 is not implemented. Otherwise, direct accesses to EDCIDSR are RES0.

Implemented only if the OPTIONAL PC Sample-based Profiling Extension is implemented in the external debug registers space.

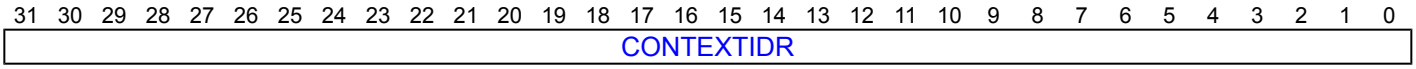
Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors registers space.

Attributes

EDCIDSR is a 32-bit register.

Field descriptions



CONTEXTIDR, bits [31:0]

Context ID. The value of CONTEXTIDR that is associated with the most recent EDPCSR sample. When the most recent EDPCSR sample is generated:

- If EL1 is using AArch64, then the Context ID is sampled from CONTEXTIDR_EL1.
- If EL1 is using AArch32, then the Context ID is sampled from CONTEXTIDR.
- If EL3 is implemented and is using AArch32, then CONTEXTIDR is a banked register, and EDCIDSR samples the current banked copy of CONTEXTIDR for the Security state that is associated with the most recent EDPCSR sample.

Because the value written to EDCIDSR is an indirect read of CONTEXTIDR, it is CONSTRAINED UNPREDICTABLE whether EDCIDSR is set to the original or new value if EDPCSR samples:

- An instruction that writes to CONTEXTIDR.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing EDCIDSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

EDCIDSR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x0A4	EDCIDSR

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDEVAFF0, External Debug Device Affinity register 0

The EDDEVAFF0 characteristics are:

Purpose

Copy of the low half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the external debug component relates to.

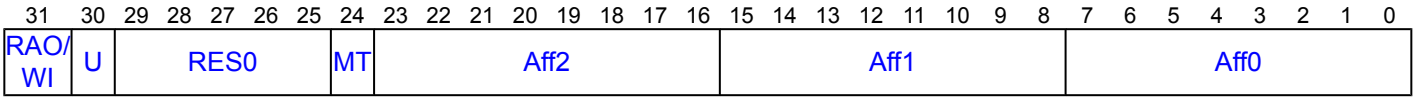
Configuration

When FEAT_DoPD is implemented, EDDEVAFF0 is in the Core power domain. Otherwise, EDDEVAFF0 is in the Debug power domain.

Attributes

EDDEVAFF0 is a 32-bit register.

Field descriptions



Bit [31]

Reserved, RAO/WI.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Access to this field is **RO**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is **RO**.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDDEVAFF0

EDDEVAFF0 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFA8	EDDEVAFF0

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDEVAFF1, External Debug Device Affinity register 1

The EDDEVAFF1 characteristics are:

Purpose

Copy of the high half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the external debug component relates to.

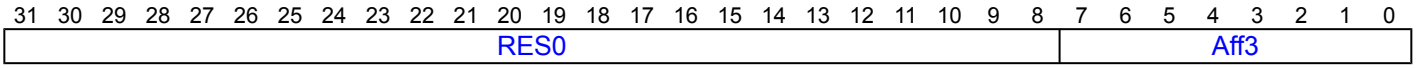
Configuration

When FEAT_DoPD is implemented, EDDEVAFF1 is in the Core power domain. Otherwise, EDDEVAFF1 is in the Debug power domain.

Attributes

EDDEVAFF1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

Aff3, bits [7:0]

Affinity level 3. See the description of EDDEVAFF0.Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDDEVAFF1

EDDEVAFF1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFAC	EDDEVAFF1

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDEVARCH, External Debug Device Architecture Register

The EDDEVARCH characteristics are:

Purpose

Identifies the programmers' model architecture of the external debug component.

Configuration

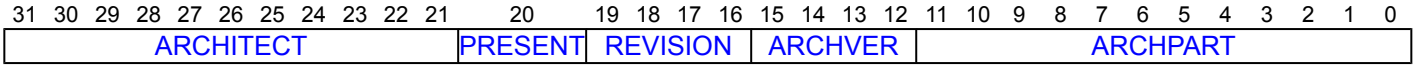
When FEAT_DoPD is implemented, EDDEVARCH is in the Core power domain. Otherwise, EDDEVARCH is in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

EDDEVARCH is a 32-bit register.

Field descriptions



ARCHITECT, bits [31:21]

Defines the architect of the component. For External Debug, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, `0b0100`.

Bits [27:21] are the JEP106 identification code, `0b0111011`.

Reads as `0b01000111011`.

Access to this field is **RO**.

PRESENT, bit [20]

DEVARCH present. Indicates that the EDDEVARCH register is present.

Reads as `0b1`.

Access to this field is **RO**.

REVISION, bits [19:16]

Defines the architecture revision. For architectures defined by Arm this is the minor revision.

For debug, the revision defined by Armv8 is `0x0`.

All other values are reserved.

Reads as `0b0000`.

Access to this field is **RO**.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ARCHVER	Meaning
0b0110	Armv8 debug architecture.
0b0111	Armv8.1 debug architecture, FEAT_Debugv8p1.
0b1000	Armv8.2 debug architecture, FEAT_Debugv8p2.
0b1001	Armv8.4 debug architecture, FEAT_Debugv8p4.
0b1010	Armv8.8 debug architecture, FEAT_Debugv8p8.
0b1011	Armv8.9 debug architecture, FEAT_Debugv8p9.

EDDEVARCH.ARCHVER and EDDEVARCH.ARCHPART are also defined as a single field, EDDEVARCH.ARCHID, so that EDDEVARCH.ARCHVER is EDDEVARCH.ARCHID[15:12].

FEAT_Debugv8p1 implements the functionality identified by the value 0b0111.

FEAT_Debugv8p2 implements the functionality identified by the value 0b1000.

FEAT_Debugv8p4 implements the functionality identified by the value 0b1001.

FEAT_Debugv8p8 implements the functionality identified by the value 0b1010.

FEAT_Debugv8p9 implements the functionality identified by the value 0b1011.

From Armv8.1, when FEAT_Debugv8p1 is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

From Armv8.9, the value 0b1010 is not permitted.

Access to this field is **RO**.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

ARCHPART	Meaning
0xA15	Armv8-A debug architecture.

EDDEVARCH.ARCHVER and EDDEVARCH.ARCHPART are also defined as a single field, EDDEVARCH.ARCHID, so that EDDEVARCH.ARCHPART is EDDEVARCH.ARCHID[11:0].

Armv8-A debug architecture.

Access to this field is **RO**.

Accessing EDDEVARCH

EDDEVARCH can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFBC	EDDEVARCH

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDEVID, External Debug Device ID register 0

The EDDEVID characteristics are:

Purpose

Provides extra information for external debuggers about features of the debug implementation.

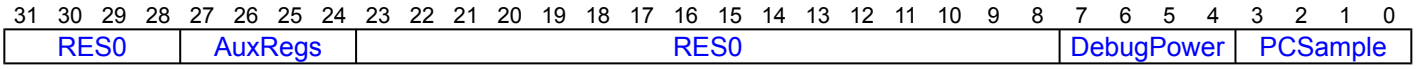
Configuration

When FEAT_DoPD is implemented, EDDEVID is in the Core power domain. Otherwise, EDDEVID is in the Debug power domain.

Attributes

EDDEVID is a 32-bit register.

Field descriptions



Bits [31:28]

Reserved, RES0.

AuxRegs, bits [27:24]

Indicates support for Auxiliary registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AuxRegs	Meaning
0b0000	None supported.
0b0001	Support for External Debug Auxiliary Control Register, EDACR.

All other values are reserved.

Access to this field is **RO**.

Bits [23:8]

Reserved, RES0.

DebugPower, bits [7:4]

Indicates support for the FEAT_DoPD feature.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DebugPower	Meaning
0b0000	FEAT_DoPD not implemented. Registers in the external debug interface register map are implemented in a mix of the Debug and Core power domains.
0b0001	FEAT_DoPD implemented. All registers in the external debug interface register map are implemented in the Core power domain.

FEAT_DoPD implements the functionality added by the value 0b0001.

All other values are reserved.

Access to this field is **RO**.

PCSample, bits [3:0]

Indicates the level of PC Sample-based Profiling support using external debug registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PCSample	Meaning
0b0000	PC Sample-based Profiling Extension is not implemented in the external debug registers space.
0b0010	Only EDPCSR and EDCIDSR are implemented. This option is only permitted if EL3 and EL2 are not implemented.
0b0011	EDPCSR, EDCIDSR, and EDVIDSR are implemented.

All other values are reserved.

When FEAT_PCSRv8p2 is implemented, the only permitted value is 0b0000.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of PMDEVID.PCSample.

Access to this field is **RO**.

Accessing EDDEVID

EDDEVID can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFC8	EDDEVID

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDEVID1, External Debug Device ID Register 1

The EDDEVID1 characteristics are:

Purpose

Provides extra information for external debuggers about features of the debug implementation.

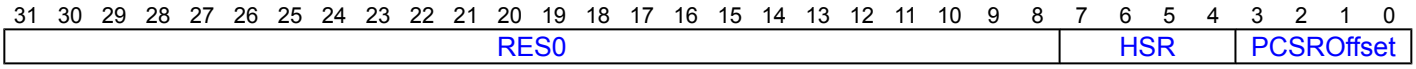
Configuration

When FEAT_DoPD is implemented, EDDEVID1 is in the Core power domain. Otherwise, EDDEVID1 is in the Debug power domain.

Attributes

EDDEVID1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

HSR, bits [7:4]

Indicates support for the External Debug Halt Status Register, EDHSR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HSR	Meaning
0b0000	EDHSR not implemented, and the PE follows behaviors consistent with all of the EDHSR fields having a zero value.
0b0001	EDHSR implemented.
0b0010	As 0b0001, but extends EDHSR to include the VNCR, CM, and WnR fields.

All other values are reserved.

FEAT_EDHSR implements the functionality identified by the value 0b0001.

FEAT_Debugv8p9 implements the functionality identified by the value 0b0010.

When FEAT_Debugv8p2 is not implemented, the only permitted value is 0b0000.

From Armv8.9, the values 0b0000 and 0b0001 are not permitted.

Access to this field is **RO**.

PCSROffset, bits [3:0]

Indicates the offset applied to PC samples returned by reads of EDPCSR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PCSROffset	Meaning
0b0000	EDPCSR not implemented.
0b0010	EDPCSR implemented, and samples have no offset applied and do not sample the instruction set state in AArch32 state.

When FEAT_PCSRv8p2 is implemented, the only permitted value is 0b0000.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of PMDEVID.PCSample.

Access to this field is **RO**.

Accessing EDDEVID1

EDDEVID1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFC4	EDDEVID1

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDEVID2, External Debug Device ID register 2

The EDDEVID2 characteristics are:

Purpose

Reserved for future descriptions of features of the debug implementation.

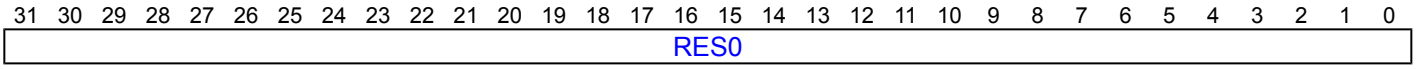
Configuration

When FEAT_DoPD is implemented, EDDEVID2 is in the Core power domain. Otherwise, EDDEVID2 is in the Debug power domain.

Attributes

EDDEVID2 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing EDDEVID2

EDDEVID2 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFC0	EDDEVID2

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDEVTYPE, External Debug Device Type register

The EDDEVTYPE characteristics are:

Purpose

Indicates to a debugger that this component is part of a PE's debug logic.

Configuration

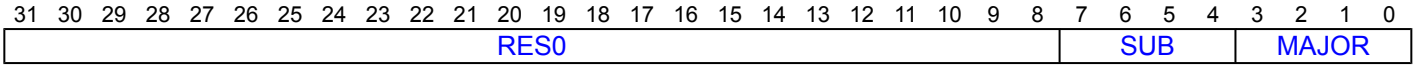
When FEAT_DoPD is implemented, EDDEVTYPE is in the Core power domain. Otherwise, EDDEVTYPE is in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

EDDEVTYPE is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Subtype. Indicates this is a component within a PE.

Reads as `0b0001`.

Access to this field is **RO**.

MAJOR, bits [3:0]

Major type. Indicates this is a debug logic component.

Reads as `0b0101`.

Access to this field is **RO**.

Accessing EDDEVTYPE

EDDEVTYPE can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0xFCC`	EDDEVTYPE

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDFR, External Debug Feature Register

The EDDFR characteristics are:

Purpose

Provides top-level information about the debug system.

Note

Debuggers must use EDDEVARCH to determine the Debug architecture version.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

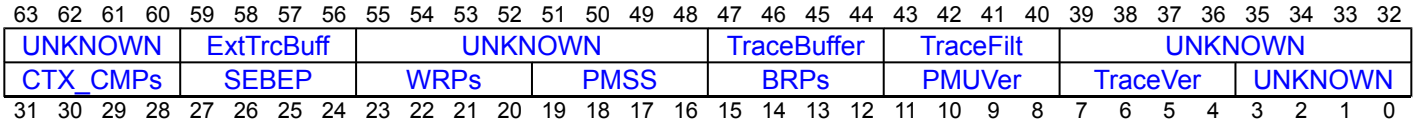
Configuration

There are no configuration notes.

Attributes

EDDFR is a 64-bit register.

Field descriptions



Bits [63:60]

Reserved, UNKNOWN.

ExtTrcBuff, bits [59:56]

Trace Buffer External Mode Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ExtTrcBuff	Meaning
0b0000	Trace Buffer Extension not implemented or Trace Buffer External Mode not implemented.
0b0001	Trace Buffer Extension implemented and Trace Buffer External Mode implemented.

All other values are reserved.

If FEAT_TRBE is not implemented, the only permitted value is 0b0000.

FEAT_TRBE_EXT implements the functionality identified by the value 0b0001.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR0_EL1.ExtTrcBuff.

Access to this field is **RO**.

Bits [55:48]

Reserved, UNKNOWN.

TraceBuffer, bits [47:44]

When FEAT_TRBE_EXT is implemented:

Trace Buffer Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceBuffer	Meaning
0b0000	Trace Buffer Extension not implemented.
0b0001	Trace Buffer Extension implemented.
0b0010	As 0b0001, and adds: • If EL2 and FEAT_FGT are implemented, a fine-grained trap on the TSB CSYNC instruction. • If EL2 is implemented, an EL2 control to override TRBLIMITR_EL1.nVM. • The TRBE Profiling exception extension, FEAT_TRBE_EXC.

All other values are reserved.

FEAT_TRBE implements the functionality identified by the value 0b0001.

FEAT_TRBEv1p1 implements the functionality identified by the value 0b0010.

In any Armv9 implementation, if FEAT_ETE is implemented, the value 0b0000 is not permitted.

From Armv9.6, if FEAT_TRBE is implemented, the value 0b0001 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, UNKNOWN.

TraceFilt, bits [43:40]

Armv8.4 Self-hosted Trace Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceFilt	Meaning
0b0000	Armv8.4 Self-hosted Trace Extension not implemented.
0b0001	Armv8.4 Self-hosted Trace Extension implemented.

All other values are reserved.

FEAT_TRF implements the functionality identified by the value 0b0001.

From Armv8.4, if FEAT_ETMv4 is implemented, the value 0b0000 is not permitted.

If FEAT_ETE is implemented, the value 0b0000 is not permitted.

Access to this field is **RO**.

Bits [39:32]

Reserved, UNKNOWN.

CTX_CMPs, bits [31:28]

Number of context-aware breakpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CTX_CMPs	Meaning
0b0000..0b1111	The number of context-aware breakpoints, minus 1.

The value of this field is never greater than EDDFR.BRPs.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR0_EL1.CTX_CMPs.

If FEAT_Debugv8p9 is implemented and 16 or more context-aware breakpoints are implemented, then this field reads as 0b1111 and EDDFR1.CTX_CMPs indicates the number of context-aware breakpoints.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 breakpoints.

Access to this field is **RO**.

SEBEP, bits [27:24]

This field either has the same value as ID_AA64DFR0_EL1.SEBEP or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

WRPs, bits [23:20]

Number of watchpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WRPs	Meaning
0b0001..0b1111	The number of watchpoints, minus 1.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR0_EL1.WRPs.

If FEAT_Debugv8p9 is implemented and 16 or more watchpoints are implemented, then this field reads as 0b1111 and EDDFR1.WRPs indicates the number of watchpoints.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 watchpoints.

The value 0b0000 is reserved.

Access to this field is **RO**.

PMSS, bits [19:16]

This field either has the same value as ID_AA64DFR0_EL1.PMSS or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

BRPs, bits [15:12]

Number of breakpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BRPs	Meaning
0b0001..0b1111	The number of breakpoints, minus 1.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR0_EL1.BRPs.

If FEAT_Debugv8p9 is implemented and 16 or more breakpoints are implemented, then this field reads as 0b1111 and EDDFR1.BRPs indicates the number of breakpoints.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 breakpoints.

The value 0b0000 is reserved.

Access to this field is **RO**.

PMUVer, bits [11:8]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version'

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMUVer	Meaning
0b0000	Performance Monitors Extension not implemented.
0b0001	Performance Monitors Extension, PMUv3 implemented.
0b0100	PMUv3 for Armv8.1. As 0b0001, and adds support for: • Extended 16-bit PMEVTYPER<n>_EL0.evtCount field. • If EL2 is implemented, the MDCR_EL2.HPMD control.
0b0101	PMUv3 for Armv8.4. As 0b0100, and adds support for the PMMIR_EL1 register.
0b0110	PMUv3 for Armv8.5. As 0b0101, and adds support for: • 64-bit event counters. • If EL2 is implemented, the MDCR_EL2.HCCD control. • If EL3 is implemented, the MDCR_EL3.SCCD control.
0b0111	PMUv3 for Armv8.7. As 0b0110, and adds support for: • The PMCR_EL0.FZO and, if EL2 is implemented, MDCR_EL2.HPMFZO controls. • If EL3 is implemented, the MDCR_EL3.{MPMX,MCCD} controls.
0b1000	PMUv3 for Armv8.8. As 0b0111, and: • Extends the Common event number space to include 0x0040 to 0x00BF and 0x4040 to 0x40BF. • Removes the CONSTRAINED UNPREDICTABLE behaviors if a reserved or unimplemented PMU event number is selected.
0b1001	PMUv3 for Armv8.9. As 0b1000, and: • Updates the definitions of existing PMU events. • Adds support for the PMUSERENR_EL0.UEN control and the PMUACR_EL1 register. • Adds support for the EDECR.PME control.
0b1111	IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value for new implementations.

All other values are reserved.

FEAT_PMUv3 implements the functionality identified by the value 0b0001.

FEAT_PMUv3p1 implements the functionality identified by the value 0b0100.

FEAT_PMUv3p4 implements the functionality identified by the value 0b0101.

FEAT_PMUv3p5 implements the functionality identified by the value 0b0110.

FEAT_PMUv3p7 implements the functionality identified by the value 0b0111.

FEAT_PMUv3p8 implements the functionality identified by the value 0b1000.

FEAT_PMUv3p9 implements the functionality identified by the value 0b1001.

From Armv8.1, if FEAT_PMUv3 is implemented, the value 0b0001 is not permitted.

From Armv8.4, if FEAT_PMUv3 is implemented, the value 0b0100 is not permitted.

From Armv8.5, if FEAT_PMUv3 is implemented, the value 0b0101 is not permitted.

From Armv8.7, if FEAT_PMUv3 is implemented, the value 0b0110 is not permitted.

From Armv8.8, if FEAT_PMUv3 is implemented, the value 0b0111 is not permitted.

From Armv8.9, if FEAT_PMUv3 is implemented, the value 0b1000 is not permitted.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR0_EL1.PMUVer.

Access to this field is **RO**.

TraceVer, bits [7:4]

Trace support. Indicates whether System register interface to a trace unit is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceVer	Meaning
0b0000	Trace unit System registers not implemented.
0b0001	Trace unit System registers implemented.

All other values are reserved.

A value of 0b0000 only indicates that no System register interface to a trace unit is implemented. A trace unit might nevertheless be implemented without a System register interface.

In an Armv8-A implementation that supports AArch64, this field returns the value of ID_AA64DFR0_EL1.TraceVer.

Access to this field is **RO**.

Bits [3:0]

Reserved, UNKNOWN.

Accessing EDDFR

EDDFR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xD28	EDDFR

Accessible as follows:

- When IsCorePowered() and !DoubleLockStatus(), accesses to this register are **RO**.
- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **IMPDEF**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDFR1, External Debug Feature Register 1

The EDDFR1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch64.

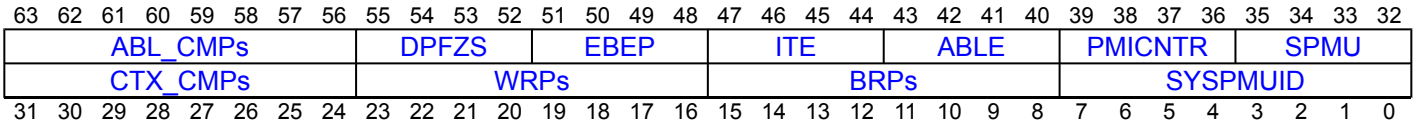
Configuration

There are no configuration notes.

Attributes

EDDFR1 is a 64-bit register.

Field descriptions



ABL_CMPs, bits [63:56]

When FEAT_ABLE is implemented:

Number of breakpoints that support address linking, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ABL_CMPs	Meaning
0x00..0x3F	Number of breakpoints that support address linking minus 1.

All other values are reserved.

The number of breakpoints that support address linking is never more than either the number of breakpoints or the number of watchpoints.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR1_EL1.ABL_CMPs.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

DPFZS, bits [55:52]

This field either has the same value as ID_AA64DFR1_EL1.DPFZS or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

EBEP, bits [51:48]

This field either has the same value as ID_AA64DFR1_EL1.EBEP or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ITE, bits [47:44]

This field either has the same value as ID_AA64DFR1_EL1.ITE or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ABLE, bits [43:40]

Address Breakpoint Linking Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ABLE	Meaning
0b0000	Address Breakpoint Linking Extension not implemented.
0b0001	Address Breakpoint Linking Extension implemented.

All other values are reserved.

FEAT_BWE implements the address range breakpoints and mismatch breakpoints part of the functionality identified by the value 0b0001.

FEAT_ABLE implements the functionality identified by the value 0b0001.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR1_EL1.ABLE.

Access to this field is **RO**.

PMICNTR, bits [39:36]

This field either has the same value as ID_AA64DFR1_EL1.PMICNTR or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SPMU, bits [35:32]

This field either has the same value as ID_AA64DFR1_EL1.SPMU or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CTX_CMPs, bits [31:24]

Context-aware breakpoints.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CTX_CMPs	Meaning
0x00	EDDFR.CTX_CMPs is the number of context-aware breakpoints, minus 1.
0x01..0x3F	Number of context-aware breakpoints minus 1.

All other values are reserved.

The value of this field is never greater than EDDFR1.BRPs.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR1_EL1.CTX_CMPs.

Access to this field is **RO**.

WRPs, bits [23:16]

Watchpoints.

WRPs	Meaning
0x00	EDDFR.WRPs is the number of watchpoints, minus 1.
0x01..0x3F	Number of watchpoints minus 1.

All other values are reserved.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR1_EL1.WRPs.

BRPs, bits [15:8]

Breakpoints.

BRPs	Meaning
0x00	EDDFR.BRPs is the number of breakpoints, minus 1.
0x01..0x3F	Number of breakpoints minus 1.

All other values are reserved.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR1_EL1.BRPs.

SYSPMUID, bits [7:0]

This field either has the same value as ID_AA64DFR1_EL1.SYSPMUID or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDDFR1

EDDFR1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xD48	EDDFR1

Accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), and (!IsZero(EDDFR1) or !OSLockStatus()), accesses to this register are **RO**.
- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **IMPDEF**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDFR2, External Debug Feature Register 2

The EDDFR2 characteristics are:

Purpose

Provides top-level information about the debug system in AArch64.

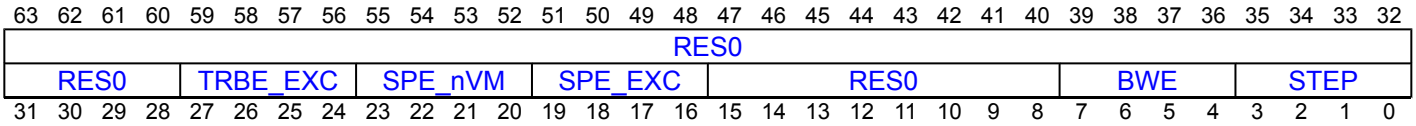
Configuration

There are no configuration notes.

Attributes

EDDFR2 is a 64-bit register.

Field descriptions



Bits [63:28]

Reserved, RES0.

TRBE_EXC, bits [27:24]

This field either has the same value as ID_AA64DFR2_EL1.TRBE_EXC or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SPE_nVM, bits [23:20]

This field either has the same value as ID_AA64DFR2_EL1.SPE_nVM or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SPE_EXC, bits [19:16]

This field either has the same value as ID_AA64DFR2_EL1.SPE_EXC or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [15:8]

Reserved, RES0.

BWE, bits [7:4]

Breakpoints and watchpoint enhancements.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BWE	Meaning
0b0000	This field does not indicate whether DBGBCR<n>_EL1.MASK and address mismatch breakpoints are implemented.
0b0001	DBGBCR<n>_EL1.MASK and address mismatch breakpoints are implemented.
0b0010	As 0b0001, and address mismatch watchpoints are implemented.

All other values are reserved.

FEAT_BWE implements the functionality identified by the value 0b0001.

FEAT_BWE2 implements the functionality identified by the value 0b0010.

When this field is 0b0000, ID_AA64DFR1_EL1.ABLE might indicate the presence of support for DBGBCR<n>_EL1.MASK and address mismatch breakpoints.

From Armv9.5, the value 0b0001 is not permitted.

In an implementation that supports AArch64, this field has the same value as ID_AA64DFR2_EL1.BWE.

Access to this field is **RO**.

STEP, bits [3:0]

This field either has the same value as ID_AA64DFR2_EL1.STEP or reads as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDDFR2

EDDFR2 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xD50	EDDFR2

Accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), and (!IsZero(EDDFR2) or !OSLockStatus()), accesses to this register are **RO**.
- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **IMPDEF**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDECCR, External Debug Exception Catch Control Register

The EDECCR characteristics are:

Purpose

Controls Exception Catch debug events. For more information, see 'Exception Catch debug event'.

Configuration

External register EDECCR bits [31:0] are architecturally mapped to AArch64 System register OSECCR_EL1[31:0].

External register EDECCR bits [31:0] are architecturally mapped to AArch32 System register DBGOSECCR[31:0].

EDECCR is in the Core power domain.

Attributes

EDECCR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3
RES0									RLR2	RLR1	RLR0	RES0	RLE2	RLE1	RLE0	NSR3	NSR2	NSR1	NSR0	SR3	SR2	SR1	SR0	NSE3	NSE2	NSE1	NSE0	SE

Bits [31:23]

Reserved, RES0.

RLR2, bit [22]

When FEAT_RME is implemented:

Controls exception catch on exception return to Realm EL2 in conjunction with EDECCR.RLE2.

RLR2	Meaning
0b0	If EDECCR.RLE2 is 0, then Exception Catch debug events are disabled for Realm EL2. If EDECCR.RLE2 is 1, then Exception Catch debug events are enabled for exception entry and exception return to Realm EL2.
0b1	If EDECCR.RLE2 is 0, then Exception Catch debug events are enabled for exception returns to Realm EL2. If EDECCR.RLE2 is 1, then Exception Catch debug events are enabled for exception entry to Realm EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

RLR1, bit [21]

When FEAT_RME is implemented:

Controls exception catch on exception return to Realm EL1 in conjunction with EDECCR.RLE1.

RLR1	Meaning
0b0	If EDECCR.RLE1 is 0, then Exception Catch debug events are disabled for Realm EL1. If EDECCR.RLE1 is 1, then Exception Catch debug events are enabled for exception entry and exception return to Realm EL1.
0b1	If EDECCR.RLE1 is 0, then Exception Catch debug events are enabled for exception returns to Realm EL1. If EDECCR.RLE1 is 1, then Exception Catch debug events are enabled for exception entry to Realm EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

RLR0, bit [20]

When FEAT_RME is implemented:

Controls exception catch on exception return to Realm EL0.

RLR0	Meaning
0b0	Exception Catch debug events are disabled for Realm EL0.
0b1	Exception Catch debug events are enabled for exception returns to Realm EL0.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [19]

Reserved, RES0.

RLE2, bit [18]

When FEAT_RME is implemented:

Controls exception catch on exception entry to Realm EL2. Also controls exception catch on exception return to Realm EL2 in conjunction with EDECCR.RLR2.

RLE2	Meaning
0b0	If EDECCR.RLR2 is 0, then Exception Catch debug events are disabled for Realm EL2. If EDECCR.RLR2 is 1, then Exception Catch debug events are enabled for exception returns to Realm EL2.
0b1	If EDECCR.RLR2 is 0, then Exception Catch debug events are enabled for exception entry and exception return to Realm EL2. If EDECCR.RLR2 is 1, then Exception Catch debug events are enabled for exception entry to Realm EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

RLE1, bit [17]

When FEAT_RME is implemented:

Controls exception catch on exception entry to Realm EL1. Also controls exception catch on exception return to Realm EL1 in conjunction with EDECCR.RLR1.

RLE1	Meaning
0b0	If EDECCR.RLR1 is 0, then Exception Catch debug events are disabled for Realm EL1. If EDECCR.RLR1 is 1, then Exception Catch debug events are enabled for exception returns to Realm EL1.
0b1	If EDECCR.RLR1 is 0, then Exception Catch debug events are enabled for exception entry and exception return to Realm EL1. If EDECCR.RLR1 is 1, then Exception Catch debug events are enabled for exception entry to Realm EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

RLE0, bit [16]

Access to this field is **RES0**.

NSR3, bit [15]

Access to this field is **RES0**.

NSR2, bit [14]

When FEAT_Debugv8p2 is implemented and Non-secure EL2 is implemented:

Controls exception catch on exception return to Non-secure EL2 in conjunction with EDECCR.NSE2.

NSR2	Meaning
0b0	If EDECCR.NSE2 is 0, then Exception Catch debug events are disabled for Non-secure EL2. If EDECCR.NSE2 is 1, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Non-secure EL2.
0b1	If EDECCR.NSE2 is 0, then Exception Catch debug events are enabled for exception returns to Non-secure EL2. If EDECCR.NSE2 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

NSR1, bit [13]

When FEAT_Debugv8p2 is implemented and Non-secure EL1 is implemented:

Controls exception catch on exception return to Non-secure EL1 in conjunction with EDECCR.NSE1.

NSR1	Meaning
0b0	If EDECCR.NSE1 is 0, then Exception Catch debug events are disabled for Non-secure EL1. If EDECCR.NSE1 is 1, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Non-secure EL1.
0b1	If EDECCR.NSE1 is 0, then Exception Catch debug events are enabled for exception returns to Non-secure EL1. If EDECCR.NSE1 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

NSR0, bit [12]

When FEAT_Debugv8p2 is implemented and Non-secure EL0 is implemented:

Controls exception catch on exception return to Non-secure EL0.

NSR0	Meaning
0b0	Exception Catch debug events are disabled for Non-secure EL0.
0b1	Exception Catch debug events are enabled for exception returns to Non-secure EL0.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SR3, bit [11]

When FEAT_Debugv8p2 is implemented and EL3 is implemented:

Controls exception catch on exception return to EL3 in conjunction with EDECCR.SE3.

SR3	Meaning
0b0	If EDECCR.SE3 is 0, then Exception Catch debug events are disabled for EL3. If EDECCR.SE3 is 1, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to EL3.
0b1	If EDECCR.SE3 is 0, then Exception Catch debug events are enabled for exception returns to EL3. If EDECCR.SE3 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to EL3.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SR2, bit [10]

When FEAT_Debugv8p2 is implemented and FEAT_SEL2 is implemented:

Controls exception catch on exception return to Secure EL2 in conjunction with EDECCR.SE2.

SR2	Meaning
0b0	If EDECCR.SE2 is 0, then Exception Catch debug events are disabled for Secure EL2. If EDECCR.SE2 is 1, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Secure EL2.
0b1	If EDECCR.SE2 is 0, then Exception Catch debug events are enabled for exception returns to Secure EL2. If EDECCR.SE2 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SR1, bit [9]

When FEAT_Debugv8p2 is implemented and Secure EL1 is implemented:

Controls exception catch on exception return to Secure EL1 in conjunction with EDECCR.SE1.

SR1	Meaning
0b0	If EDECCR.SE1 is 0, then Exception Catch debug events are disabled for Secure EL1. If EDECCR.SE1 is 1, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Secure EL1.
0b1	If EDECCR.SE1 is 0, then Exception Catch debug events are enabled for exception returns to Secure EL1. If EDECCR.SE1 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SR0, bit [8]

When FEAT_Debugv8p2 is implemented and Secure EL0 is implemented:

Controls exception catch on exception return to Secure EL0.

SR0	Meaning
0b0	Exception Catch debug events are disabled for Secure EL0.
0b1	Exception Catch debug events are enabled for exception returns to Secure EL0.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

NSE3, bit [7]

Access to this field is **RES0**.

NSE2, bit [6]

When FEAT_Debugv8p2 is implemented and Non-secure EL2 is implemented:

Controls exception catch on exception entry to Non-secure EL2. Also controls exception catch on exception return to Non-secure EL2 in conjunction with EDECCR.NSR2.

NSE2	Meaning
0b0	If EDECCR.NSR2 is 0, then Exception Catch debug events are disabled for Non-secure EL2. If EDECCR.NSR2 is 1, then Exception Catch debug events are enabled for exception returns to Non-secure EL2.
0b1	If EDECCR.NSR2 is 0, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Non-secure EL2. If EDECCR.NSR2 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure EL2.

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level will generate an Exception Catch debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

When Non-secure EL2 is implemented:

Coarse-grained exception catch for Non-secure EL2. Controls Exception Catch debug events for Non-secure EL2.

NSE2	Meaning
0b0	Exception Catch debug events are disabled for Non-secure EL2.
0b1	Exception Catch debug events are enabled for Non-secure EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

NSE1, bit [5]

When FEAT_Debugv8p2 is implemented and Non-secure EL1 is implemented:

Controls exception catch on exception entry to Non-secure EL1. Also controls exception catch on exception return to Non-secure EL1 in conjunction with EDECCR.NSR1.

NSE1	Meaning
0b0	If EDECCR.NSR1 is 0, then Exception Catch debug events are disabled for Non-secure EL1. If EDECCR.NSR1 is 1, then Exception Catch debug events are enabled for exception returns to Non-secure EL1.
0b1	If EDECCR.NSR1 is 0, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Non-secure EL1. If EDECCR.NSR1 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure EL1.

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level will generate an Exception Catch debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

When Non-secure EL1 is implemented:

Coarse-grained exception catch for Non-secure EL1. Controls Exception Catch debug events for Non-secure EL1.

NSE1	Meaning
0b0	Exception Catch debug events are disabled for Non-secure EL1.
0b1	Exception Catch debug events are enabled for Non-secure EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

NSE0, bit [4]

Access to this field is **RES0**.

SE3, bit [3]

When FEAT_Debugv8p2 is implemented and EL3 is implemented:

Controls exception catch on exception entry to EL3. Also controls exception catch on exception return to EL3 in conjunction with EDECCR.SR3.

SE3	Meaning
0b0	If EDECCR.SR3 is 0, then Exception Catch debug events are disabled for EL3. If EDECCR.SR3 is 1, then Exception Catch debug events are enabled for exception returns to EL3.
0b1	If EDECCR.SR3 is 0, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to EL3. If EDECCR.SR3 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to EL3.

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level will generate an Exception Catch debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

When FEAT_Debugv8p2 is not implemented and EL3 is implemented:

Coarse-grained exception catch for EL3. Controls Exception Catch debug events for EL3.

SE3	Meaning
0b0	Exception Catch debug events are disabled for EL3.
0b1	Exception Catch debug events are enabled for EL3.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SE2, bit [2]

When FEAT_Debugv8p2 is implemented and FEAT_SEL2 is implemented:

Controls exception catch on exception entry to Secure EL2. Also controls exception catch on exception return to Secure EL2 in conjunction with EDECCR.SR2.

SE2	Meaning
0b0	If EDECCR.SR2 is 0, then Exception Catch debug events are disabled for Secure EL2. If EDECCR.SR2 is 1, then Exception Catch debug events are enabled for exception returns to Secure EL2.
0b1	If EDECCR.SR2 is 0, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Secure EL2. If EDECCR.SR2 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure EL2.

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level will generate an Exception Catch debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SE1, bit [1]

When FEAT_Debugv8p2 is implemented and Secure EL1 is implemented:

Controls exception catch on exception entry to Secure EL1. Also controls exception catch on exception return to Secure EL1 in conjunction with EDECCR.SR1.

SE1	Meaning
0b0	If EDECCR.SR1 is 0, then Exception Catch debug events are disabled for Secure EL1. If EDECCR.SR1 is 1, then Exception Catch debug events are enabled for exception returns to Secure EL1.
0b1	If EDECCR.SR1 is 0, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Secure EL1. If EDECCR.SR1 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure EL1.

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level will generate an Exception Catch debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

When Secure EL1 is implemented:

Coarse-grained exception catch for Secure EL1. Controls Exception Catch debug events for Secure EL1.

SE1	Meaning
0b0	Exception Catch debug events are disabled for Secure EL1.
0b1	Exception Catch debug events are enabled for Secure EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SE0, bit [0]

Access to this field is **RES0**.

Accessing EDECCR

EDECCR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x098	EDECCR

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDECR, External Debug Execution Control Register

The EDECR characteristics are:

Purpose

Controls Halting debug events.

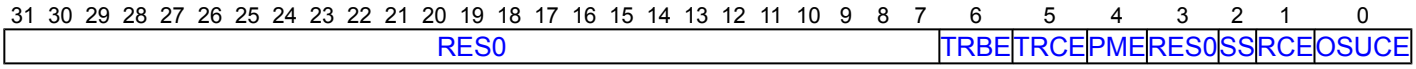
Configuration

When FEAT_DoPD is implemented, EDECR is in the Core power domain. Otherwise, EDECR is in the Debug power domain.

Attributes

EDECR is a 32-bit register.

Field descriptions



Bits [31:7]

Reserved, RES0.

TRBE, bit [6]

When FEAT_Debugv8p9 is implemented and FEAT_TRBE_EXT is implemented:

Trace Buffer External Debug Request Enable.

TRBE	Meaning
0b0	Trace Buffer External Debug Request disabled.
0b1	Trace Buffer External Debug Request enabled.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

TRCE, bit [5]

When FEAT_ETEv1p3 is implemented and FEAT_Debugv8p9 is implemented:

ETE External Debug Request Enable.

TRCE	Meaning
0b0	ETE External Debug Request disabled.
0b1	ETE External Debug Request enabled.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

PME, bit [4]

When FEAT_Debugv8p9 is implemented and FEAT_PMUv3p9 is implemented:

PMU Overflow External Debug Request Enable.

PME	Meaning
0b0	PMU Overflow External Debug Request disabled.
0b1	PMU Overflow External Debug Request enabled.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [3]

Reserved, RES0.

SS, bit [2]

Halting step enable. Possible values of this field are:

SS	Meaning
0b0	Halting step debug event disabled.
0b1	Halting step debug event enabled.

If the value of EDECR.SS is changed when the PE is in Non-debug state, behavior is CONSTRAINED UNPREDICTABLE as described in 'Changing the value of EDECR.SS when not in Debug state'.

The reset behavior of this field is:

- On a Cold reset, when FEAT_DoPD is implemented, this field resets to '0'.
- On an External debug reset, when FEAT_DoPD is not implemented, this field resets to '0'.

RCE, bit [1]

When FEAT_DoPD is not implemented:

Reset Catch Enable.

RCE	Meaning
0b0	Reset Catch debug event disabled.
0b1	Reset Catch debug event enabled.

The reset behavior of this field is:

- On an External debug reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

OSUCE, bit [0]
When FEAT_DoPD is not implemented:

OS Unlock Catch Enable.

OSUCE	Meaning
0b0	OS Unlock Catch debug event disabled.
0b1	OS Unlock Catch debug event enabled.

The reset behavior of this field is:

- On an External debug reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Accessing EDECR

EDECR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x024	EDECR

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDESR, External Debug Event Status Register

The EDESR characteristics are:

Purpose

Indicates the status of internally pending Halting debug events.

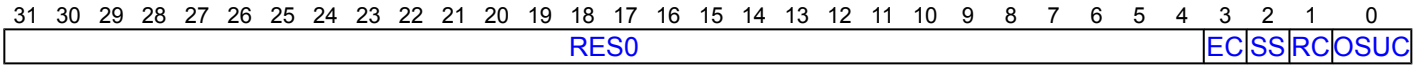
Configuration

EDESR is in the Core power domain.

Attributes

EDESR is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

EC, bit [3]

When FEAT_Debugv8p8 is implemented:

Exception Catch debug event pending.

EC	Meaning
0b0	Exception Catch debug event is not pending.
0b1	Exception Catch debug event is pending.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

SS, bit [2]

When FEAT_DoPD is implemented:

Halting step debug event pending. Possible values of this field are:

SS	Meaning
0b0	Reading this means that a Halting step debug event is not pending. Writing this means no action.
0b1	Reading this means that a Halting step debug event is pending. Writing this clears the pending Halting step debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Halting step debug event pending. Possible values of this field are:

SS	Meaning
0b0	Reading this means that a Halting step debug event is not pending. Writing this means no action.
0b1	Reading this means that a Halting step debug event is pending. Writing this clears the pending Halting step debug event.

The reset behavior of this field is:

- On a Warm reset, this field resets to the value in EDECR.SS.

RC, bit [1]

Reset Catch debug event pending. Possible values of this field are:

RC	Meaning
0b0	Reading this means that a Reset Catch debug event is not pending. Writing this means no action.
0b1	Reading this means that a Reset Catch debug event is pending. Writing this clears the pending Reset Catch debug event.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_DoPD is implemented, this field resets to the value in CTIDEVCTL.RCE.
  - When FEAT_DoPD is not implemented, this field resets to the value in EDECR.RCE.

OSUC, bit [0]

OS Unlock Catch debug event pending. Possible values of this field are:

OSUC	Meaning
0b0	Reading this means that an OS Unlock Catch debug event is not pending. Writing this means no action.
0b1	Reading this means that an OS Unlock Catch debug event is pending. Writing this clears the pending OS Unlock Catch debug event.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing EDESR

If a request to clear a pending Halting debug event is received at or about the time when halting becomes allowed, it is CONSTRAINED UNPREDICTABLE whether the event is taken.

If Core power is removed while a Halting debug event is pending, it is lost. However, it might become pending again when the Core is powered back on and Cold reset.

EDESR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x020	EDESR

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDHSR, External Debug Halting Syndrome Register

The EDHSR characteristics are:

Purpose

Holds syndrome information for a debug event.

Configuration

EDHSR is in the Core power domain.

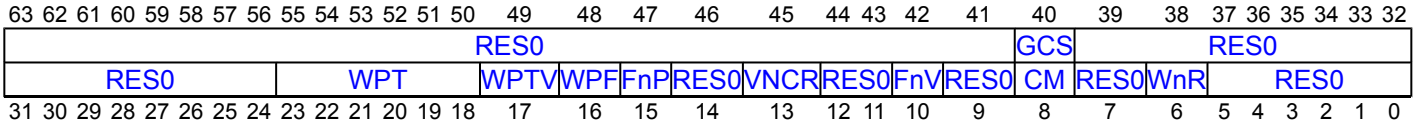
This register is present only when FEAT_EDHSR is implemented. Otherwise, direct accesses to EDHSR are RES0.

The value of this register is UNKNOWN if the PE is in Non-debug state, or if EDSCR.STATUS is not `0b101011`.

Attributes

EDHSR is a 64-bit register.

Field descriptions



Bits [63:41]

Reserved, RES0.

GCS, bit [40]

When FEAT_GCS is implemented and FEAT_Debugv8p9 is implemented:

Guarded control stack data access.

Indicates that the Watchpoint debug event is due to a Guarded control stack data access.

GCS	Meaning
`0b0`	The Watchpoint debug event is not due to a Guarded control stack data access.
`0b1`	The Watchpoint debug event is due to a Guarded control stack data access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [39:24]

Reserved, RES0.

WPT, bits [23:18]

Watchpoint number. When EDHSR.WPTV is 1, holds the index of a watchpoint that triggered the Watchpoint debug event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WPTV, bit [17]

Watchpoint number valid.

WPTV	Meaning	Applies when
0b0	EDHSR.WPT field is not valid, and holds an UNKNOWN value.	When FEAT_Debugv8p9 is not implemented
0b1	EDHSR.WPT field is valid, and holds the number of a watchpoint that triggered the Watchpoint debug event.	

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WPF, bit [16]

Watchpoint might be false-positive.

WPF	Meaning	Applies when
0b0	The watchpoint matched an address or address range that was accessed by the instruction.	
0b1	The watchpoint matched an address or address range that might not have been accessed by the instruction.	When FEAT_SVE is implemented or FEAT_SME is implemented

Arm strongly recommends that this bit is set to 0, other than when one of the following instructions might generate a watchpoint match for an address or address range that the instruction does not access:

- An SVE contiguous vector load/store instruction, when the PE is in Streaming SVE mode.
- An SME load/store instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FnP, bit [15]

EDWAR not Precise.

FnP	Meaning	Applies when
0b0	If the EDWAR is valid, it holds the virtual address of an access or sequence of contiguous accesses that triggered the Watchpoint debug event.	
0b1	If the EDWAR is valid, it holds any virtual address within the smallest implemented translation granule that contains the virtual address of an access or set of contiguous accesses that triggered the Watchpoint debug event.	When FEAT_SME is implemented or FEAT_SVE is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [14]

Reserved, RES0.

VNCR, bit [13]

When FEAT_Debugv8p9 is implemented:

VNCR_EL2 access. Indicates that the Watchpoint debug event came from use of VNCR_EL2 register by EL1 code.

VNCR	Meaning	Applies when
0b0	The Watchpoint debug event was not generated by the use of VNCR_EL2 by EL1 code.	
0b1	The Watchpoint debug event was generated by the use of VNCR_EL2 by EL1 code.	When FEAT_NV2 is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:11]

Reserved, RES0.

FnV, bit [10]

EDWAR not Valid.

FnV	Meaning	Applies when
0b0	EDWAR is valid.	
0b1	EDWAR is not valid, and holds an UNKNOWN value.	When FEAT_SME is implemented or FEAT_SVE is implemented

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

CM, bit [8]

When FEAT_Debugv8p9 is implemented:

Cache maintenance. Indicates whether the Watchpoint debug event came from a cache maintenance instruction.

CM	Meaning
0b0	The Watchpoint debug event was not generated by the execution of one of the System instructions identified in the description of value 1.
0b1	The Watchpoint debug event was generated by the execution of a cache maintenance instruction. The DC ZVA, DC GVA, and DC GZVA instructions are not cache maintenance instructions, and therefore do not cause this field to be set to 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

WnR, bit [6]

When FEAT_Debugv8p9 is implemented:

Write not Read. Indicates whether the Watchpoint debug event was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

WnR	Meaning
0b0	Watchpoint debug event caused by an instruction reading from a memory location.
0b1	Watchpoint debug event caused by an instruction writing to a memory location.

For Watchpoint debug events on cache maintenance instructions, this field is set to 1.

For Watchpoint debug events from an atomic instruction, this field is set to 0 if a read of the location would have generated the Watchpoint debug event, otherwise it is set to 1.

If multiple watchpoints match on the same access, it is UNPREDICTABLE which watchpoint generates the Watchpoint debug event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [5:0]

Reserved, RES0.

Accessing EDHSR

EDHSR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x038	EDHSR

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDITCTRL, External Debug Integration mode Control register

The EDITCTRL characteristics are:

Purpose

Enables the external debug to switch from its default mode into integration mode, where test software can control directly the inputs and outputs of the PE, for integration testing or topology detection.

Configuration

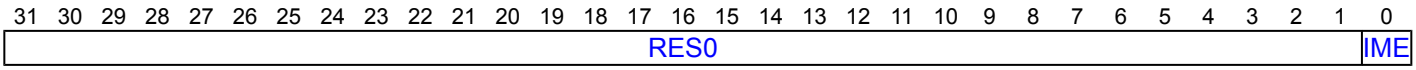
The power domain of EDITCTRL is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

Attributes

EDITCTRL is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

IME, bit [0]

Integration mode enable. When IME == 1, the device reverts to an integration mode to enable integration testing or topology detection.

IME	Meaning
0b0	Normal operation.
0b1	Integration mode enabled.

The integration mode behavior is IMPLEMENTATION DEFINED.

The following resets apply:

- Whichever power domain the register is implemented in, this field resets to 0.
- Otherwise, the value of this field is unchanged.

Accessing EDITCTRL

EDITCTRL can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xF00	EDITCTRL

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register are **IMPDEF**.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDITR, External Debug Instruction Transfer Register

The EDITR characteristics are:

Purpose

Used in Debug state for passing instructions to the PE for execution.

Configuration

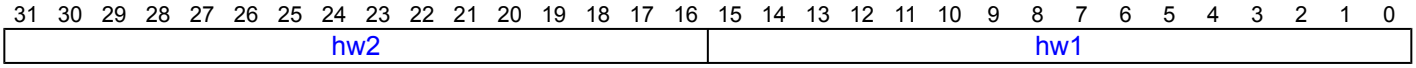
EDITR is in the Core power domain.

Attributes

EDITR is a 32-bit register.

Field descriptions

When FEAT_AA32 is implemented and in AArch32 state:



hw2, bits [31:16]

Second halfword of the T32 instruction to be executed on the PE. When EDITR contains a 16-bit T32 instruction, this field is ignored. For more information, see 'Behavior in Debug state'.

Note

The hw2 field is displayed on the left. This is not the usual convention for display of T32 instruction halfwords.

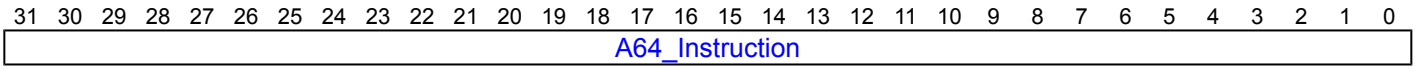
hw1, bits [15:0]

First halfword of the T32 instruction to be executed on the PE.

Note

The hw1 field is displayed on the right. This is not the usual convention for display of T32 instruction halfwords.

When FEAT_AA64 is implemented and in AArch64 state:



A64_Instruction, bits [31:0]

A64 instruction to be executed on the PE.

Accessing EDITR

If EDSCR.ITE == 0 when the PE exits Debug state on receiving a Restart request trigger event, the behavior of any instruction issued through the ITR in Normal access mode that has not completed execution is CONSTRAINED UNPREDICTABLE, and must do one of the following:

- It must complete execution in Debug state before the PE executes the restart sequence.
- It must complete execution in Non-debug state before the PE executes the restart sequence.
- It must be abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an UNKNOWN state.

EDITR ignores writes if the PE is in Non-debug state.

EDITR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x084	EDITR

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **WI**.
- Otherwise, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDLAR, External Debug Lock Access Register

The EDLAR characteristics are:

Purpose

Allows or disallows access to the external debug registers through a memory-mapped interface.

The optional Software Lock provides a lock to prevent memory-mapped writes to the debug registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the debug registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

When FEAT_DoPD is implemented, EDLAR is in the Core power domain. Otherwise, EDLAR is in the Debug power domain.

If FEAT_DoPD is implemented, Software Lock is not implemented by the architecturally-defined debug components of the PE.

Software uses EDLAR to set or clear the lock, and EDLSR to check the current status of the lock.

Attributes

EDLAR is a 32-bit register.

Field descriptions

When Debug Software Lock is implemented:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KEY																															

KEY, bits [31:0]

Lock Access control. Writing the key value `0xC5ACCE55` to this field unlocks the lock, enabling write accesses to this component's registers through a memory-mapped interface.

Writing any other value to this register locks the lock, disabling write accesses to this component's registers through a memory mapped interface.

Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0																															

Otherwise

Bits [31:0]

Reserved, RES0.

Accessing EDLAR

EDLAR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
Debug	`0xFB0`	EDLAR

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDLSR, External Debug Lock Status Register

The EDLSR characteristics are:

Purpose

Indicates the current status of the software lock for external debug registers.

The optional Software Lock provides a lock to prevent memory-mapped writes to the debug registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the debug registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

When FEAT_DoPD is implemented, EDLSR is in the Core power domain. Otherwise, EDLSR is in the Debug power domain.

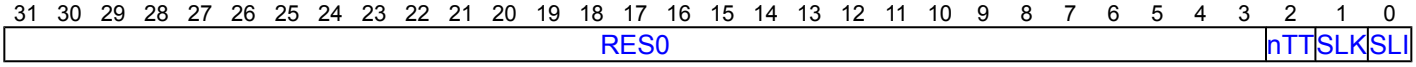
If FEAT_DoPD is implemented, Software Lock is not implemented by the architecturally-defined debug components of the PE.

Software uses EDLAR to set or clear the lock, and EDLSR to check the current status of the lock.

Attributes

EDLSR is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

nTT, bit [2]

Not thirty-two bit access required. RAZ.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SLK, bit [1]

When Debug Software Lock is implemented:

Software Lock status for this component. For an access to LSR that is not a memory-mapped access, or when Software Lock is not implemented, this field is RES0.

For memory-mapped accesses when Software Lock is implemented, possible values of this field are:

SLK	Meaning
0b0	Lock clear. Writes are permitted to this component's registers.
0b1	Lock set. Writes to this component's registers are ignored, and reads have no side effects.

The reset behavior of this field is:

- On a Cold reset, when FEAT_DoPD is implemented, this field resets to '1'.
- On an External debug reset, when FEAT_DoPD is not implemented, this field resets to '1'.

Otherwise:

Reserved, RAZ.

SLI, bit [0]

Software Lock implemented. For an access to LSR that is not a memory-mapped access, this field is RAZ.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SLI	Meaning
0b0	Software Lock not implemented or not memory-mapped access.
0b1	Software Lock implemented and memory-mapped access.

Access to this field is **RO**.

Accessing EDLSR

EDLSR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
Debug	0xFB4	EDLSR

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPCSR, External Debug Program Counter Sample Register

The EDPCSR characteristics are:

Purpose

Holds a sampled instruction address value.

Configuration

EDPCSR is in the Core power domain.

This register is present only when FEAT_PCSRv8 is implemented and FEAT_PCSRv8p2 is not implemented. Otherwise, direct accesses to EDPCSR are RES0.

EDPCSR[63:32] and EDPCSR[31:0] are accessed at 32-bit memory mapped addresses that are not contiguous.

If FEAT_Debugv8p1 is implemented, the format of this register differs depending on the value of EDSCR.SC2.

Implemented only if the OPTIONAL PC Sample-based Profiling Extension is implemented in the external debug registers space.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors registers space.

Attributes

EDPCSR is a 64-bit register.

Field descriptions

When FEAT_Debugv8p1 is not implemented or EDSCR.SC2 == 0:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
EDPCSRhi																															
EDPCSRlo																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

EDPCSRhi, bits [63:32]

PC Sample high word, EDPCSRhi. If EDVIDSR.HV == 0 then this field is RAZ, otherwise bits [63:32] of the sampled instruction address value. The translation regime that EDPCSR samples can be determined from EDVIDSR.{NS,E2,E3}.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

EDPCSRlo, bits [31:0]

PC Sample low word. EDPCSRlo, bits[31:0] of the sampled instruction address value.

EDPCSRlo reads as `0xFFFFFFFF` when any of the following are true:

- The PE is in Debug state.
- PC Sample-based profiling is prohibited.

If a branch instruction has retired since the PE left reset state, then the first read of EDPCSR[31:0] is permitted but not required to return `0xFFFFFFFF`.

EDPCSRlo reads as an UNKNOWN value when any of the following are true:

- The PE is in reset state.
- No branch instruction has retired since the PE left reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No branch instruction has retired since the last read of EDPCSR[31:0].

For the cases where a read of EDPCSR[31:0] returns `0xFFFFFFFF` or an UNKNOWN value, the read has the side-effect of setting EDPCSRhi, EDCIDSR, and EDVIDSR to UNKNOWN values.

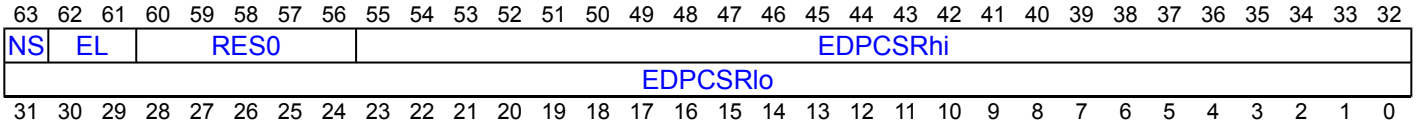
Otherwise, a read of EDPCSR[31:0] returns bits [31:0] of the sampled instruction address value and has the side-effect of indirectly writing to EDPCSRhi, EDCIDSR, and EDVIDSR. The translation regime that EDPCSR samples can be determined from EDVIDSR.{NS,E2,E3}.

For a read of EDPCSR[31:0] from the memory-mapped interface, if EDLSR.SLK == 1, meaning the OPTIONAL Software Lock is locked, then the side-effect of the access does not occur and EDPCSRhi, EDCIDSR, and EDVIDSR are unchanged.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When FEAT_Debugv8p1 is implemented and EDSCR.SC2 == 1:



NS, bit [63]

Non-secure state sample. Indicates the Security state that is associated with the most recent EDPCSR sample or, when it is read as a single atomic 64-bit read, the current EDPCSR sample. The translation regime that EDPCSR samples can be determined from EDPCSR.{NS,EL}.

If EL3 is not implemented, this bit indicates the Effective value of SCR.NS.

NS	Meaning
`0b0`	Sample is from Secure state.
`0b1`	Sample is from Non-secure state.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

EL, bits [62:61]

Exception level status sample. Indicates the Exception level that is associated with the most recent EDPCSR sample or, when it is read as a single atomic 64-bit read, the current EDPCSR sample. The translation regime that EDPCSR samples can be determined from EDPCSR.{NS,EL}.

EL	Meaning
`0b00`	Sample is from EL0.
`0b01`	Sample is from EL1.
`0b10`	Sample is from EL2.
`0b11`	Sample is from EL3.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [60:56]

Reserved, RES0.

EDPCSRhi, bits [55:32]

PC Sample high word, EDPCSRhi. Bits [55:32] of the sampled instruction address value. The translation regime that EDPCSR samples can be determined from EDPCSR.{NS,EL}.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

EDPCSRlo, bits [31:0]

PC Sample low word. EDPCSRlo, bits[31:0] of the sampled instruction address value.

EDPCSRlo reads as `0xFFFFFFFF` when any of the following are true:

- The PE is in Debug state.
- PC Sample-based profiling is prohibited.

If a branch instruction has retired since the PE left reset state, then the first read of EDPCSR[31:0] is permitted but not required to return `0xFFFFFFFF`.

EDPCSRlo reads as an UNKNOWN value when any of the following are true:

- The PE is in reset state.
- No branch instruction has retired since the PE left reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No branch instruction has retired since the last read of EDPCSR[31:0].

For the cases where a read of EDPCSR[31:0] returns `0xFFFFFFFF` or an UNKNOWN value, the read has the side-effect of setting EDPCSRhi, EDCIDSR, and EDVIDSR to UNKNOWN values.

Otherwise, a read of EDPCSR[31:0] returns bits [31:0] of the sampled instruction address value and has the side-effect of indirectly writing to EDPCSRhi, EDCIDSR, and EDVIDSR. The translation regime that EDPCSR samples can be determined from EDPCSR.{NS,EL}.

For a read of EDPCSR[31:0] from the memory-mapped interface, if EDLSR.SLK == 1, meaning the OPTIONAL Software Lock is locked, then the side-effect of the access does not occur and EDPCSRhi, EDCIDSR, and EDVIDSR are unchanged.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing EDPCSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'

EDPCSR can be accessed through the external debug interface:

Component	Offset	Instance	Range
Debug	0x0A0	EDPCSR	31:0

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Component	Offset	Instance	Range
Debug	0x0AC	EDPCSR	63:32

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPFR, External Debug Processor Feature Register

The EDPFR characteristics are:

Purpose

Provides information about implemented PE features.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

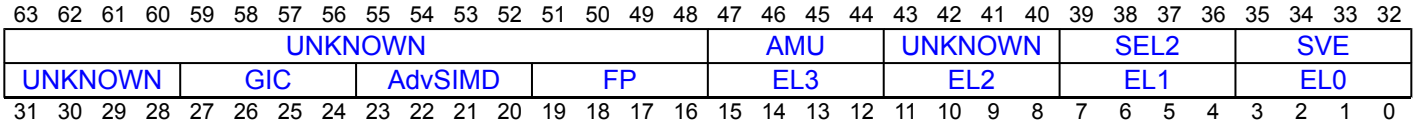
Configuration

There are no configuration notes.

Attributes

EDPFR is a 64-bit register.

Field descriptions



Bits [63:48]

Reserved, UNKNOWN.

AMU, bits [47:44]

Indicates support for Activity Monitors Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AMU	Meaning
0b0000	Activity Monitors Extension is not implemented.
0b0001	FEAT_AMUv1 is implemented.
0b0010	FEAT_AMUv1p1 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters.

All other values are reserved.

FEAT_AMUv1 implements the functionality identified by the value 0b0001.

FEAT_AMUv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0000.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.6, the permitted values are 0b0000, 0b0001, and 0b0010.

Access to this field is **RO**.

Bits [43:40]

Reserved, UNKNOWN.

SEL2, bits [39:36]

Secure EL2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEL2	Meaning
0b0000	Secure EL2 is not implemented.
0b0001	Secure EL2 is implemented.

FEAT_SEL2 implements the functionality identified by the value 0b0001

All other values are reserved.

Access to this field is **RO**.

SVE, bits [35:32]

Scalable Vector Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SVE	Meaning
0b0000	SVE is not implemented.
0b0001	SVE is implemented.

FEAT_SVE implements the functionality identified by the value 0b0001.

All other values are reserved.

Access to this field is **RO**.

Bits [31:28]

Reserved, UNKNOWN.

GIC, bits [27:24]

System register GIC interface support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GIC	Meaning
0b0000	GIC CPU interface system registers not implemented.
0b0001	System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.
0b0011	System register interface to version 4.1 of the GIC CPU interface is supported.

All other values are reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of ID_AA64PFR0_EL1.GIC.

Access to this field is **RO**.

AdvSIMD, bits [23:20]

Advanced SIMD.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AdvSIMD	Meaning
0b0000	Advanced SIMD is implemented, including support for the following SISD and SIMD operations: • Integer byte, halfword, word and doubleword element operations. • Single-precision and double-precision floating-point arithmetic. • Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.
0b0001	As for 0b0000, and also includes support for half-precision floating-point arithmetic.
0b1111	Advanced SIMD is not implemented.

All other values are reserved.

This field must have the same value as the FP field.

The permitted values are:

- 0b0000 in an implementation with Advanced SIMD support, that does not include the FEAT_FP16 extension.
- 0b0001 in an implementation with Advanced SIMD support, that includes the FEAT_FP16 extension.
- 0b1111 in an implementation without Advanced SIMD support.

In an Armv8-A implementation that supports AArch64, this field returns the value of ID_AA64PFR0_EL1.AdvSIMD.

Access to this field is **RO**.

FP, bits [19:16]

Floating-point.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FP	Meaning
0b0000	Floating-point is implemented, and includes support for: • Single-precision and double-precision floating-point types. • Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.
0b0001	As for 0b0000, and also includes support for half-precision floating-point arithmetic.
0b1111	Floating-point is not implemented.

All other values are reserved.

This field must have the same value as the AdvSIMD field.

The permitted values are:

- 0b0000 in an implementation with floating-point support, that does not include the FEAT_FP16 extension.
- 0b0001 in an implementation with floating-point support, that includes the FEAT_FP16 extension.
- 0b1111 in an implementation without floating-point support.

In an Armv8-A implementation that supports AArch64, this field returns the value of ID_AA64PFR0_EL1.FP.

Access to this field is **RO**.

EL3, bits [15:12]

AArch64 EL3 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL3	Meaning
0b0000	EL3 is not implemented or cannot be executed in AArch64 state.
0b0001	EL3 can be executed in AArch64 state only.
0b0010	EL3 can be executed in both Execution states.

When the value of EDAA32PFR.EL3 is nonzero, this field must be 0b0000.

All other values are reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of ID_AA64PFR0_EL1.EL3.

Access to this field is **RO**.

EL2, bits [11:8]

AArch64 EL2 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL2	Meaning
0b0000	EL2 is not implemented or cannot be executed in AArch64 state.
0b0001	EL2 can be executed in AArch64 state only.
0b0010	EL2 can be executed in both Execution states.

When the value of EDAA32PFR.EL2 is nonzero, this field must be 0b0000.

All other values are reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of ID_AA64PFR0_EL1.EL2.

Access to this field is **RO**.

EL1, bits [7:4]

AArch64 EL1 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL1	Meaning
0b0000	EL1 cannot be executed in AArch64 state. EL1 can be executed in AArch32 state only.
0b0001	EL1 can be executed in AArch64 state only.
0b0010	EL1 can be executed in both Execution states.

All other values are reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of ID_AA64PFR0_EL1.EL1.

Access to this field is **RO**.

EL0, bits [3:0]

AArch64 EL0 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL0	Meaning
0b0000	EL0 cannot be executed in AArch64 state. EL0 can be executed in AArch32 state only.
0b0001	EL0 can be executed in AArch64 state only.
0b0010	EL0 can be executed in both Execution states.

All other values are reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of ID_AA64PFR0_EL1.EL0.

Access to this field is **RO**.

Accessing EDPFR

EDPFR can be accessed through the external debug interface:

Component	Offset	Instance

Debug	0xD20	EDPFR

Accessible as follows:

- When IsCorePowered() and !DoubleLockStatus(), accesses to this register are **RO**.
- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **IMPDEF**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPIDR0, External Debug Peripheral Identification Register 0

The EDPIDR0 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

When FEAT_DoPD is implemented, EDPIDR0 is in the Core power domain. Otherwise, EDPIDR0 is in the Debug power domain.

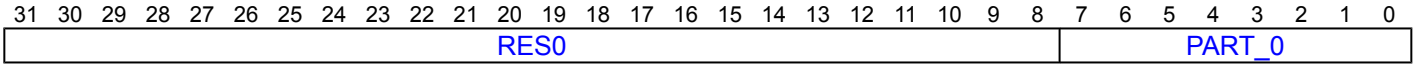
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

EDPIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, least significant byte.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDPIDR0

EDPIDR0 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFE0	EDPIDR0

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPIDR1, External Debug Peripheral Identification Register 1

The EDPIDR1 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

When FEAT_DoPD is implemented, EDPIDR1 is in the Core power domain. Otherwise, EDPIDR1 is in the Debug power domain.

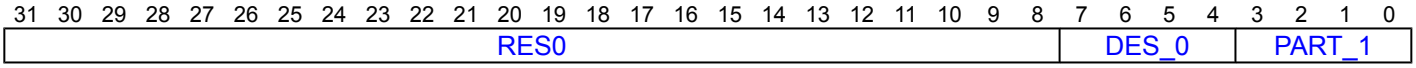
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

EDPIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, least significant nibble of JEP106 ID code. For Arm Limited, this field is `0b1011`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, most significant nibble.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDPIDR1

EDPIDR1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0xFE4`	EDPIDR1

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPIDR2, External Debug Peripheral Identification Register 2

The EDPIDR2 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

When FEAT_DoPD is implemented, EDPIDR2 is in the Core power domain. Otherwise, EDPIDR2 is in the Debug power domain.

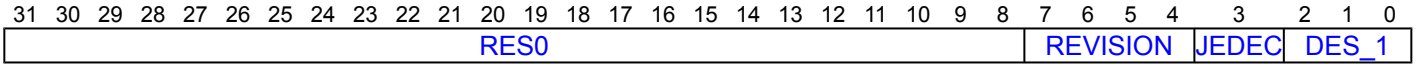
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

EDPIDR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Part major revision. Parts can also use this field to extend Part number to 16-bits.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

Indicates a JEP106 identity code is used.

Reads as `0b1`.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, most significant bits of JEP106 ID code. For Arm Limited, this field is `0b011`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDPIDR2

EDPIDR2 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFE8	EDPIDR2

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPIDR3, External Debug Peripheral Identification Register 3

The EDPIDR3 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

When FEAT_DoPD is implemented, EDPIDR3 is in the Core power domain. Otherwise, EDPIDR3 is in the Debug power domain.

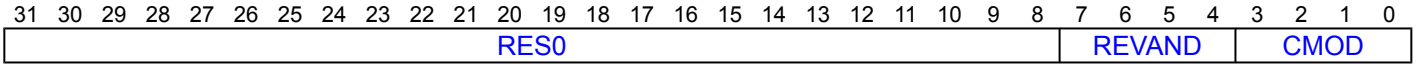
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

EDPIDR3 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Part minor revision. Parts using EDPIDR2.REVISION as an extension to the Part number must use this field as a major revision number.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer modified. Indicates someone other than the Designer has modified the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDPIDR3

EDPIDR3 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xFEC	EDPIDR3

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPIDR4, External Debug Peripheral Identification Register 4

The EDPIDR4 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

When FEAT_DoPD is implemented, EDPIDR4 is in the Core power domain. Otherwise, EDPIDR4 is in the Debug power domain.

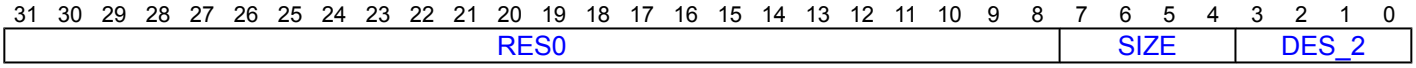
Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

EDPIDR4 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component. Log_2 of the number of 4KB pages from the start of the component to the end of the component ID registers.

Reads as `0b0000`.

Access to this field is **RO**.

DES_2, bits [3:0]

Designer, JEP106 continuation code, least significant nibble. For Arm Limited, this field is `0b0100`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDPIDR4

EDPIDR4 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0xFD0`	EDPIDR4

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPRCR, External Debug Power/Reset Control Register

The EDPRCR characteristics are:

Purpose

Controls the PE functionality related to powerup, reset, and powerdown.

Configuration

External register EDPRCR bit [0] is architecturally mapped to AArch64 System register DBGPRCR_EL1[0].

External register EDPRCR bit [0] is architecturally mapped to AArch32 System register DBGPRCR[0].

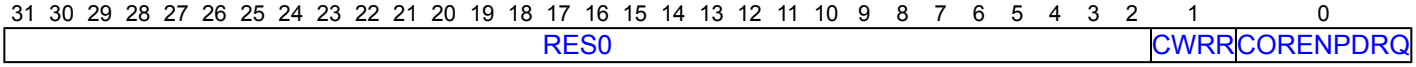
When FEAT_DoPD is implemented, EDPRCR is in the Core power domain. Otherwise, EDPRCR contains fields that are in the Core power domain and fields that are in the Debug power domain.

Attributes

EDPRCR is a 32-bit register.

Field descriptions

When FEAT_DoPD is implemented:



Bits [31:2]

Reserved, RES0.

CWRR, bit [1]

When FEAT_RME is implemented:

The PE ignores all writes to this bit.

Otherwise:

Warm reset request.

The extent of the reset is IMPLEMENTATION DEFINED, but must be one of:

- The request is ignored.
- Only this PE is Warm reset.
- This PE and other components of the system, possibly including other PEs, are Warm reset.

Arm deprecates use of this bit, and recommends that implementations ignore the request.

CWRR	Meaning
0b0	No action.
0b1	Request Warm reset.

This field is in the Core power domain

The PE ignores writes to this bit if any of the following are true:

- ExternalInvasiveDebugEnabled() == FALSE, EL3 is not implemented, and the implemented Security state is Non-secure state.
- ExternalSecureInvasiveDebugEnabled() == FALSE, EL3 is not implemented, and the implemented Security state is Secure state.
- ExternalSecureInvasiveDebugEnabled() == FALSE and EL3 is implemented.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGRSTREQ** signal.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if any the following are true:
  - OSLockStatus().
  - SoftwareLockStatus().
- Otherwise, access to this field is **WO/RAZ**.

CORENPDRQ, bit [0]

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

CORENPDRQ	Meaning
`0b0`	If the system responds to a powerdown request, it powers down Core power domain.
`0b1`	If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.

When this bit reads as UNKNOWN, the PE ignores writes to this bit.

This field is in the Core power domain, and permitted accesses to this field map to the DBGPRCR.CORENPDRQ and DBGPRCR_EL1.CORENPDRQ fields.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGNOPWRDWN** signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the Cold reset value on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states, see 'Core power domain power states'.

Note

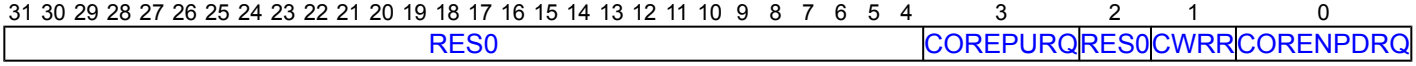
Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

On a Cold reset, if the powerup request is implemented and the powerup request has been asserted, this field is an IMPLEMENTATION DEFINED choice of 0 or 1. If the powerup request is not asserted, this field is set to 0.

Accessing this field has the following behavior:

- When OSLockStatus(), access to this field is **UNKNOWN/WI**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Otherwise:



Bits [31:4]

Reserved, RES0.

COREPURQ, bit [3]

Core powerup request. Allows a debugger to request that the power controller power up the core, enabling access to the debug register in the Core power domain, and that the power controller emulates powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

COREPURQ	Meaning
0b0	Do not request power up of the Core power domain.
0b1	Request power up of the Core power domain, and emulation of powerdown.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGPWRUPREQ** signal.

This field is in the Debug power domain and can be read and written when the Core power domain is powered off.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The reset behavior of this field is:

- On an External debug reset, this field resets to '0'.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Bit [2]

Reserved, RES0.

CWRR, bit [1]

When FEAT_RME is implemented:

The PE ignores all writes to this bit.

Otherwise:

Warm reset request.

The extent of the reset is IMPLEMENTATION DEFINED, but must be one of:

- The request is ignored.
- Only this PE is Warm reset.
- This PE and other components of the system, possibly including other PEs, are Warm reset.

Arm deprecates use of this bit, and recommends that implementations ignore the request.

CWRR	Meaning
0b0	No action.
0b1	Request Warm reset.

This field is in the Core power domain

The PE ignores writes to this bit if any of the following are true:

- ExternalInvasiveDebugEnabled() == FALSE, EL3 is not implemented, and the implemented Security state is Non-secure state.
- ExternalSecureInvasiveDebugEnabled() == FALSE, EL3 is not implemented, and the implemented Security state is Secure state.
- ExternalSecureInvasiveDebugEnabled() == FALSE and EL3 is implemented.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGRSTREQ** signal.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if any the following are true:
  - !IsCorePowered().
  - DoubleLockStatus().
  - OSLockStatus().
  - SoftwareLockStatus().
- Otherwise, access to this field is **WO/RAZ**.

CORENPDRQ, bit [0]

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

CORENPDRQ	Meaning
`0b0`	If the system responds to a powerdown request, it powers down Core power domain.
`0b1`	If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.

When this bit reads as UNKNOWN, the PE ignores writes to this bit.

This field is in the Core power domain, and permitted accesses to this field map to the DBGPRCR.CORENPDRQ and DBGPRCR_EL1.CORENPDRQ fields.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGNOPWRDWN** signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the value of EDPRCR.COREPURQ on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states, see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The reset behavior of this field is:

- On a Cold reset, this field resets to the value in EDPRCR.COREPURQ.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - !IsCorePowered().
  - DoubleLockStatus().
  - OSLockStatus().
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Accessing EDPRCR

On permitted accesses to the register, other access controls affect the behavior of some fields. See the field descriptions for more information.

EDPRCR can be accessed through the external debug interface:

Component	Offset	Instance

Debug	0x310	EDPRCR

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPRSR, External Debug Processor Status Register

The EDPRSR characteristics are:

Purpose

Holds information about the reset and powerdown state of the PE.

Configuration

When FEAT_DoPD is implemented, EDPRSR is in the Core power domain. Otherwise, EDPRSR contains fields that are in the Core power domain and fields that are in the Debug power domain.

If FEAT_DoPD is implemented then all fields in this register are in the Core power domain.

Attributes

EDPRSR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0															EPMADE	ETADE	EDADE	STAD	ETAD	SDR	SPMAD	EPMAD	SDAD	EDAD	DLK	OSLK	HALTED	SR	R	SP	

Bits [31:17]

Reserved, RES0.

EPMADE, bit [16]

When FEAT_RME is implemented and FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable Extended status. Together with EDPRSR.EPMAD, reports whether access to Performance Monitor registers by an external debugger is prohibited by the MDCR_EL3.{EPMAD, EPMADE} controls.

For a description of the values derived by evaluating EDPRSR.EPMAD and EDPRSR.EPMADE together, see EDPRSR.EPMAD.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - EDPRSR.R == 1.
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

ETADE, bit [15]

When FEAT_RME is implemented, FEAT_TRC_EXT is implemented, and FEAT_TRBE is implemented:

External Trace Access Disable Extended status. Together with EDPRSR.ETAD, reports whether access to trace unit registers by an external debugger is prohibited by the MDCR_EL3.{ETAD, ETADE} controls.

For a description of the values derived by evaluating EDPRSR.ETAD and EDPRSR.ETADE together, see EDPRSR.ETAD.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - EDPRSR.R == 1.
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

EDADE, bit [14]

When FEAT_RME is implemented:

External Debug Access Disable Extended status. Together with EDPRSR.EDAD, reports whether access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger is prohibited by the MDCR_EL3.{EDAD, EDADE} controls.

For a description of the values derived by evaluating EDPRSR.EDAD and EDPRSR.EDADE together, see EDPRSR.EDAD.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - EDPRSR.R == 1.
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

STAD, bit [13]

When FEAT_TRC_EXT is implemented and FEAT_TRBE is implemented:

Sticky ETAD error. Set to 1 when a Non-secure external debug interface access to an external trace register returns an error because `AllowExternalTraceAccess()` == FALSE for the access.

STAD	Meaning
`0b0`	Since EDPRSR was last read, no external accesses to the trace unit registers have failed because `AllowExternalTraceAccess()` was FALSE for the access.
`0b1`	Since EDPRSR was last read, at least one external access to the trace unit registers has failed because `AllowExternalTraceAccess()` was FALSE for the access.

If `IsCorePowered()` == TRUE, the Core power domain is powered up, then, following a read of EDPRSR, then this bit clears to 0.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
  - FEAT_DoPD is not implemented.
  - !IsCorePowered().

- Otherwise, access to this field is **RC/WI**.

Otherwise:

Reserved, RES0.

ETAD, bit [12]

When FEAT_RME is implemented, FEAT_TRC_EXT is implemented, and FEAT_TRBE is implemented:

External Trace Access Disable status. Together with EDPRSR.ETADE, reports whether access to trace unit registers by an external debugger is prohibited by the MDCR_EL3.{ETAD, ETADE} controls.

ETADE	ETAD	Meaning
0b0	0b0	No accesses from an external debugger to trace unit registers are prohibited.
0b0	0b1	Realm and Non-secure accesses from an external debugger to trace unit registers are prohibited. Other accesses from an external debugger to trace unit registers are not affected.
0b1	0b0	Secure and Non-secure accesses from an external debugger to trace unit registers are prohibited. Other accesses from an external debugger to trace unit registers are not affected.
0b1	0b1	Secure, Non-secure, and Realm accesses from an external debugger to trace unit registers are prohibited. Other accesses from an external debugger to trace unit registers are not affected.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - EDPRSR.R == 1.
- Otherwise, access to this field is **RO**.

When FEAT_TRC_EXT is implemented and FEAT_TRBE is implemented:

External Trace Access Disable status. Reports whether Non-secure access to trace unit registers by an external debugger is prohibited by the MDCR_EL3.ETAD control.

ETAD	Meaning
0b0	External Non-secure trace unit accesses not affected. `AllowExternalTraceAccess()` == TRUE for a Non-secure access.
0b1	External Non-secure trace unit accesses are prohibited. `AllowExternalTraceAccess()` == FALSE for a Non-secure access.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - EDPRSR.R == 1.
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

SDR, bit [11]

Sticky Debug Restart. Set to 1 when the PE exits Debug state.

SDR	Meaning
0b0	The PE has not restarted since EDPRSR was last read.
0b1	The PE has restarted since EDPRSR was last read.

Note

If a reset occurs when the PE is in Debug state, the PE exits Debug state. SDR is UNKNOWN on Warm reset, meaning a debugger must also use the SR bit to determine whether the PE has left Debug state.

If the Core power domain is powered up, then following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or `DoubleLockStatus()` == FALSE this bit clears to 0.
- If FEAT_DoubleLock is implemented and `DoubleLockStatus()` == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
  - FEAT_DoPD is not implemented.
  - !IsCorePowered().
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC/WI**.

SPMAD, bit [10]

When FEAT_Debugv8p4 is implemented and FEAT_PMUv3_EXT is implemented:

Sticky EPMAD error. Set to 1 if an external debug interface access to a Performance Monitors register returns an error because `AllowExternalPMUAccess()` == FALSE.

SPMAD	Meaning
0b0	No Non-secure external debug interface accesses to the external Performance Monitors registers have failed because `AllowExternalPMUAccess()` == FALSE for the access since EDPRSR was last read.
0b1	At least one Non-secure external debug interface access to the external Performance Monitors register has failed and returned an error because `AllowExternalPMUAccess()` == FALSE for the access since EDPRSR was last read.

If the Core power domain is powered up, then following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or `DoubleLockStatus()` == FALSE, this bit clears to 0.
- If FEAT_DoubleLock is implemented and `DoubleLockStatus()` == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
  - FEAT_DoPD is not implemented.
  - !IsCorePowered().
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC/WI**.

When FEAT_PMUv3_EXT is implemented:

Sticky EPMAD error.

SPMAD	Meaning
0b0	No external debug interface accesses to the Performance Monitors registers have failed because AllowExternalPMUAccess() == FALSE since EDPRSR was last read.
0b1	At least one external debug interface access to the Performance Monitors registers has failed and returned an error because AllowExternalPMUAccess() == FALSE since EDPRSR was last read.

If the Core power domain is powered up, then, following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE, this bit clears to 0.
- If FEAT_DoubleLock is implemented, and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
  - FEAT_DoPD is not implemented.
  - !IsCorePowered().
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- When OSLockStatus(), access to this field is **UNKNOWN/WI**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC/WI**.

Otherwise:

Reserved, RES0.

EPMAD, bit [9]
When FEAT_RME is implemented and FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable status. Together with EDPRSR.EPMADE, reports whether access to Performance Monitor registers by an external debugger is prohibited by the MDCR_EL3.{EPMAD, EPMADE} controls.

See MDCR_EL3.EPMAD for the list of affected external Performance Monitor registers.

EPMADE	EPMAD	Meaning
0b0	0b0	No accesses from an external debugger to affected Performance Monitor registers are prohibited.
0b0	0b1	Realm and Non-secure accesses from an external debugger to affected Performance Monitor registers are prohibited. Other accesses from an external debugger to affected Performance Monitor registers are not affected.
0b1	0b0	Secure and Non-secure accesses from an external debugger to affected Performance Monitor registers are prohibited. Other accesses from an external debugger to affected Performance Monitor registers are not affected.
0b1	0b1	Secure, Non-secure, and Realm accesses from an external debugger to affected Performance Monitor registers are prohibited. Other accesses from an external debugger to affected Performance Monitor registers are not affected.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
    - FEAT_DoPD is not implemented and !IsCorePowered().
    - EDPRSR.R == 1.
- Otherwise, access to this field is **RO**.

When FEAT_Debugv8p4 is implemented and FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable status. Reports whether Non-secure access to Performance Monitor registers by an external debugger is prohibited by the MDCR_EL3.EPMAD control.

See MDCR_EL3.EPMAD for the list of affected external Performance Monitor registers.

EPMAD	Meaning
0b0	External Non-secure access to Performance Monitor registers not affected. `AllowExternalPMUAccess()` == TRUE for a Non-secure access.
0b1	External Non-secure access to affected Performance Monitor registers is prohibited. `AllowExternalPMUAccess()` == FALSE for a Non-secure access.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
    - FEAT_DoPD is not implemented and !IsCorePowered().
    - EDPRSR.R == 1.
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

When FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable status. Reports whether access to Performance Monitor registers by an external debugger is prohibited by the MDCR_EL3.EPMAD control.

See MDCR_EL3.EPMAD for the list of affected external Performance Monitor registers.

EPMAD	Meaning
0b0	External access to Performance Monitor registers not affected. `AllowExternalPMUAccess()` == TRUE.
0b1	External access to affected Performance Monitor registers is prohibited. `AllowExternalPMUAccess()` == FALSE.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
    - FEAT_DoPD is not implemented and !IsCorePowered().
    - EDPRSR.R == 1.
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- When OSLockStatus(), access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

When FEAT_PMUv3 is implemented:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

SDAD, bit [8]

When FEAT_Debugv8p4 is implemented:

Sticky EDAD error. Set to 1 if an external debug interface access to a debug register returns an error because `AllowExternalDebugAccess()` == FALSE.

SDAD	Meaning
`0b0`	No Non-secure external debug interface accesses to the debug registers have failed because `AllowExternalDebugAccess()` == FALSE for the access since EDPRSR was last read.
`0b1`	At least one Non-secure external debug interface access to the debug registers has failed and returned an error because `AllowExternalDebugAccess()` == FALSE for the access since EDPRSR was last read.

If the Core power domain is powered up, then, following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or `DoubleLockStatus()` == FALSE this bit clears to 0.
- If FEAT_DoubleLock is implemented and `DoubleLockStatus()` == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - EDPRSR.R == 1.
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

Otherwise:

Sticky EDAD error. Set to 1 if an external debug interface access to a debug register returns an error because `AllowExternalDebugAccess()` == FALSE.

SDAD	Meaning
`0b0`	No external debug interface accesses to the debug registers have failed because `AllowExternalDebugAccess()` == FALSE since EDPRSR was last read.
`0b1`	At least one external debug interface access to the debug registers has failed and returned an error because `AllowExternalDebugAccess()` == FALSE since EDPRSR was last read.

If the Core power domain is powered up, then, following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or `DoubleLockStatus()` == FALSE this bit clears to 0.
- If FEAT_DoubleLock is implemented and `DoubleLockStatus()` == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - EDPRSR.R == 1.
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- Access to this field is **UNKNOWN/WI** if all the following are true:
  - OSLockStatus().

    - external debug writes to OSLAR_EL1 do not return an error when AllowExternalDebugAccess(addrdesc) == FALSE.
- Otherwise, access to this field is **RO**.

EDAD, bit [7]

When FEAT_RME is implemented:

External Debug Access Disable status. Together with EDPRSR.EDADE, reports whether access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger is prohibited by the MDCR_EL3.{EDAD, EDADE} controls.

See MDCR_EL3.EDAD for the list of affected external debug registers.

EDADE	EDAD	Meaning
0b0	0b0	No accesses from an external debugger to affected debug registers are prohibited.
0b0	0b1	Realm and Non-secure accesses from an external debugger to affected debug registers are prohibited. Other accesses from an external debugger to affected debug registers are not affected.
0b1	0b0	Secure and Non-secure accesses from an external debugger to affected debug registers are prohibited. Other accesses from an external debugger to affected debug registers are not affected.
0b1	0b1	Secure, Non-secure, and Realm accesses from an external debugger to affected debug registers are prohibited. Other accesses from an external debugger to affected debug registers are not affected.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
    - FEAT_DoPD is not implemented and !IsCorePowered().
    - EDPRSR.R == 1.
- Otherwise, access to this field is **RO**.

When FEAT_Debugv8p4 is implemented:

External Debug Access Disable status. Reports whether Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger is prohibited by the MDCR_EL3.EDAD control.

See MDCR_EL3.EDAD for the list of affected external debug registers.

EDAD	Meaning
0b0	External Non-secure access to debug registers not affected. `AllowExternalDebugAccess()` == TRUE for a Non-secure access.
0b1	External Non-secure access to affected debug registers is prohibited. `AllowExternalDebugAccess()` == FALSE for a Non-secure access.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
    - FEAT_DoPD is not implemented and !IsCorePowered().
    - EDPRSR.R == 1.
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

When FEAT_Debugv8p2 is implemented:

External Debug Access Disable status. Reports whether access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger is prohibited by the MDCR_EL3.EDAD control.

See MDCR_EL3.EDAD for the list of affected external debug registers.

EDAD	Meaning
0b0	External access to debug registers not affected. AllowExternalDebugAccess() == TRUE.
0b1	External access to affected debug registers is prohibited. AllowExternalDebugAccess() == FALSE.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - EDPRSR.R == 1.
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- Access to this field is **UNKNOWN/WI** if all the following are true:
  - OSLockStatus().
  - external debug writes to OSLAR_EL1 do not return an error when AllowExternalDebugAccess(addrdesc) == FALSE.
- Otherwise, access to this field is **RO**.

Otherwise:

External Debug Access Disable status. Reports whether access to breakpoint registers, watchpoint registers, and optionally OSLAR_EL1 by an external debugger is prohibited by the MDCR_EL3.EDAD control.

See MDCR_EL3.EDAD for the list of affected external debug registers.

EDAD	Meaning
0b0	External access to debug registers not affected. AllowExternalDebugAccess() == TRUE.
0b1	External access to affected debug registers is prohibited. AllowExternalDebugAccess() == FALSE.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - EDPRSR.R == 1.
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

DLK, bit [6]

When FEAT_Debugv8p4 is implemented:

This field is RES0.

When FEAT_Debugv8p2 is implemented and FEAT_DoubleLock is implemented:

Double Lock. From Armv8.2, use of this field is deprecated.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - !IsCorePowered().
  - DoubleLockStatus().
- Otherwise, access to this field is **RAZ/WI**.

When FEAT_DoubleLock is implemented:

Double Lock.

This field returns the result of the pseudocode function `DoubleLockStatus()`.

If the Core power domain is powered up and `DoubleLockStatus()` == TRUE, it is IMPLEMENTATION DEFINED whether:

- EDPRSR.PU reads as 1, EDPRSR.DLK reads as 1, and EDPRSR.SPD is UNKNOWN.
- EDPRSR.PU reads as 0, EDPRSR.DLK is UNKNOWN, and EDPRSR.SPD reads as 0.

This field is in the Core power domain.

DLK	Meaning
0b0	`DoubleLockStatus()` returns FALSE.
0b1	`DoubleLockStatus()` returns TRUE and the Core power domain is powered up.

Accessing this field has the following behavior:

- When !IsCorePowered(), access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

OSLK, bit [5]

OS Lock status bit.

A read of this bit returns the value of OSLSR_EL1.OSLK.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - EDPRSR.R == 1.
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

HALTED, bit [4]

Halted status bit.

HALTED	Meaning
0b0	PE is in Non-debug state.
0b1	PE is in Debug state.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
  - FEAT_DoPD is not implemented.
  - !IsCorePowered().
- Otherwise, access to this field is **RO**.

SR, bit [3]

Sticky core Reset status bit.

SR	Meaning
0b0	The non-debug logic of the PE is not in reset state and has not been reset since the last time EDPRSR was read.
0b1	The non-debug logic of the PE is in reset state or has been reset since the last time EDPRSR was read.

If EDPRSR.PU reads as 1 and EDPRSR.R reads as 0, which means that the Core power domain is in a powerup state and that the non-debug logic of the PE is not in reset state, then following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or `DoubleLockStatus()` == FALSE this bit clears to 0.
- If FEAT_DoubleLock is implemented and `DoubleLockStatus()` == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'1'`.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
  - FEAT_DoPD is not implemented.
  - !IsCorePowered().
- When DoubleLockStatus(), access to this field is **UNKNOWN/WI**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC/WI**.

R, bit [2]

PE Reset status bit.

R	Meaning
0b0	The non-debug logic of the PE is not in reset state.
0b1	The non-debug logic of the PE is in reset state.

If FEAT_DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONSTRAINED UNPREDICTABLE value. For more information, see 'EDPRSR.{DLK, R} and reset state'.

This field is in the Core power domain.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - FEAT_DoPD is not implemented and !IsCorePowered().
  - DoubleLockStatus().
- Otherwise, access to this field is **RO**.

SPD, bit [1]

Sticky core Powerdown status bit.

If FEAT_DoubleLock is implemented and `DoubleLockStatus()` == TRUE, then:

- If FEAT_Debugv8p2 is implemented, this bit reads as 0.
- If FEAT_Debugv8p2 is not implemented, this bit might read as 0 or 1.

For more information, see 'EDPRSR.{DLK, SPD, PU} and the Core power domain'.

SPD	Meaning
0b0	If EDPRSR.PU is 0, it is not known whether the state of the debug registers in the Core power domain is lost. If EDPRSR.PU is 1, the state of the debug registers in the Core power domain has not been lost.
0b1	The state of the debug registers in the Core power domain has been lost.

If the Core power domain is powered up, then, following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or `DoubleLockStatus()` == FALSE this bit clears to 0.
- If FEAT_DoubleLock is implemented and `DoubleLockStatus()` == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} and the Core power domain'.

When FEAT_DoPD is not implemented and the Core power domain is in either retention or powerdown state, the value of EDPRSR.SPD is IMPLEMENTATION DEFINED. For more information, see 'EDPRSR.SPD when the Core domain is in either retention or powerdown state'.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'1'`.

Accessing this field has the following behavior:

- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_DoPD is not implemented.
  - !IsCorePowered().
- Access to this field is **UNKNOWN/WI** if all the following are true:
  - IsCorePowered().
  - DoubleLockStatus().
- Otherwise, access to this field is **RC/WI**.

PU, bit [0]

When FEAT_DoPD is implemented:

Core powerup status bit.

Access to this field is **RAO/WI**.

When FEAT_Debugv8p2 is implemented:

Core Powerup status bit. Indicates whether the debug registers in the Core power domain can be accessed.

PU	Meaning
`0b0`	Either the Core power domain is in a low-power or powerdown state, or FEAT_DoubleLock is implemented and `DoubleLockStatus()` == TRUE, meaning the debug registers in the Core power domain cannot be accessed.
`0b1`	The Core power domain is in a powerup state, and either FEAT_DoubleLock is not implemented or `DoubleLockStatus()` == FALSE, meaning the debug registers in the Core power domain can be accessed.

If FEAT_DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONSTRAINED UNPREDICTABLE value. For more information, see 'EDPRSR.{DLK, R} and reset state'

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} and the Core power domain'

Access to this field is **RO**.

Otherwise:

Core Powerup status bit. Indicates whether the debug registers in the Core power domain can be accessed.

PU	Meaning
`0b0`	Core power domain is in a low-power or powerdown state where the debug registers in the Core power domain cannot be accessed.
`0b1`	Core power domain is in a powerup state where the debug registers in the Core power domain can be accessed.

If FEAT_DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONSTRAINED UNPREDICTABLE value. For more information see 'EDPRSR.{DLK, R} and reset state'

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} and the Core power domain'.

When the Core power domain is powered-up and `DoubleLockStatus() == TRUE`, then the value of EDPRSR.PU is IMPLEMENTATION DEFINED. See the description of the DLK bit for more information.

If FEAT_DoubleLock is implemented, the Core power domain is powered up, and `DoubleLockStatus() == TRUE`, it is IMPLEMENTATION DEFINED whether this bit reads as 0 or 1.

Access to this field is **RO**.

Accessing EDPRSR

On permitted accesses to the register, other access controls affect the behavior of some fields. See the field descriptions for more information.

If the Core power domain is powered up (EDPRSR.PU == 1), then following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or `DoubleLockStatus() == FALSE`, then:
  - EDPRSR.{SDR, SPMAD, SDAD, SPD} are cleared to 0.
  - EDPRSR.SR is cleared to 0 if the non-debug logic of the PE is not in reset state (EDPRSR.R == 0).
- If FEAT_DoubleLock is implemented and `DoubleLockStatus() == TRUE`, it is CONSTRAINED UNPREDICTABLE whether or not this clearing occurs.

If FEAT_DoPD is not implemented and the Core power domain is powered down (EDPRSR.PU == 0), then:

- EDPRSR.{SDR, SPMAD, SDAD, SR} are all UNKNOWN, and are either reset or restored on being powered up.
- EDPRSR.SPD is not cleared following a read of EDPRSR. See the SPD bit description for more information.

The clearing of bits is an indirect write to EDPRSR.

EDPRSR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0x314`	EDPRSR

Accessible as follows:

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDRCR, External Debug Reserve Control Register

The EDRCR characteristics are:

Purpose

This register is used to allow imprecise entry to Debug state and clear sticky bits in EDSCR.

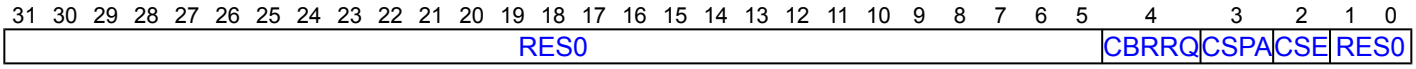
Configuration

EDRCR is in the Core power domain.

Attributes

EDRCR is a 32-bit register.

Field descriptions



Bits [31:5]

Reserved, RES0.

CBRRQ, bit [4]

Allow imprecise entry to Debug state. The actions on writing to this bit are:

CBRRQ	Meaning
0b0	No action.
0b1	Allow imprecise entry to Debug state, for example by canceling pending bus accesses.

Setting this bit to 1 allows a debugger to request imprecise entry to Debug state. An External Debug Request debug event must be pending before the debugger sets this bit to 1.

This feature is optional. If this feature is not implemented, writes to this bit are ignored.

CSPA, bit [3]

Clear Sticky Pipeline Advance. This bit is used to clear the EDSCR.PipeAdv bit to 0. The actions on writing to this bit are:

CSPA	Meaning
0b0	No action.
0b1	Clear the EDSCR.PipeAdv bit to 0.

CSE, bit [2]

Clear Sticky Error. Used to clear the EDSCR cumulative error bits to 0. The actions on writing to this bit are:

CSE	Meaning
0b0	No action.
0b1	Clear the EDSCR.{TXU, RXO, ERR} bits, and, if the PE is in Debug state, the EDSCR.ITO bit, to 0.

Bits [1:0]

Reserved, RES0.

Accessing EDRCR

EDRCR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x090	EDRCR

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **WI**.
- Otherwise, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDSCR, External Debug Status and Control Register

The EDSCR characteristics are:

Purpose

Main control register for the debug implementation.

Configuration

External register EDSCR bits [31:29, 27:26, 23:21, 19, 14, 6] are architecturally mapped to AArch64 System register MDSCR_EL1[31:29, 27:26, 23:21, 19, 14, 6].

External register EDSCR bits [31:29, 27:26, 23:21, 19, 14, 6] are architecturally mapped to AArch32 System register DBGDSCRext[31:29, 27:26, 23:21, 19, 14, 6].

External register EDSCR bits [30:29] are architecturally mapped to AArch64 System register MDCCSR_EL0[30:29].

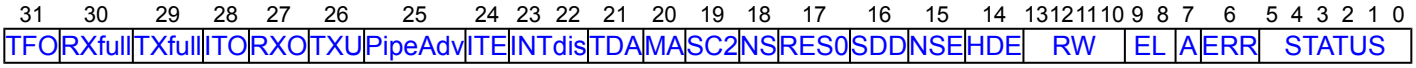
External register EDSCR bits [30:29] are architecturally mapped to AArch32 System register DBGDSCRint[30:29].

EDSCR is in the Core power domain.

Attributes

EDSCR is a 32-bit register.

Field descriptions



TFO, bit [31]

When FEAT_TRF is implemented:

Trace Filter Override. Overrides the Trace Filter controls allowing the external debugger to trace any visible Exception level.

TFO	Meaning
0b0	Trace Filter controls are not affected.
0b1	Trace Filter controls in TRFCR_EL1 and TRFCR_EL2 are ignored. Trace Filter controls TRFCR and HTRFCR are ignored.

When OSLSR_EL1.OSLK is 1, this bit can be indirectly read and written through MDSCR_EL1 and DBGDSCRext.

This bit is ignored by the PE when any of the following is true:

- ExternalSecureNoninvasiveDebugEnabled() is FALSE and the Effective value of MDCR_EL3.STE is 1.
- FEAT_RME is implemented, ExternalRealmNoninvasiveDebugEnabled() is FALSE, and the Effective value of MDCR_EL3.RLTE is 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

RXfull, bit [30]

DTRRX full.

When OSLSR_EL1.OSLK is 1, this bit can be indirectly read and written through MDSCR_EL1 and DBGDSCRext.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Access to this field is **RO**.

TXfull, bit [29]

DTRTX full.

When OSLSR_EL1.OSLK is 1, this bit can be indirectly read and written through MDSCR_EL1 and DBGDSCRext.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Access to this field is **RO**.

ITO, bit [28]

ITR overrun. Set to 0 on entry to Debug state.

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

RXO, bit [27]

DTRRX overrun.

When OSLSR_EL1.OSLK is 1, this bit can be indirectly read and written through MDSCR_EL1 and DBGDSCRext.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Access to this field is **RO**.

TXU, bit [26]

DTRTX underrun.

When OSLSR_EL1.OSLK is 1, this bit can be indirectly read and written through MDSCR_EL1 and DBGDSCRext.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Access to this field is **RO**.

PipeAdv, bit [25]

Pipeline Advance. Indicates that software execution is progressing.

PipeAdv	Meaning
`0b0`	No progress has been made by the PE since the last time this field was cleared to zero by writing 1 to EDRCR.CSPA.
`0b1`	Progress has been made by the PE since the last time this field was cleared to zero by writing 1 to EDRCR.CSPA.

The architecture does not define precisely when this field is set to 1. It requires only that this happen periodically in Non-debug state to indicate that software execution is progressing. For example, a PE might set this field to 1 each time the PE retires one or more instructions, or at periodic intervals during the progression of an instruction.

When FEAT_MOPS is implemented, `CPY`, `CPYF`, `SET`, and `SETG` are examples of instructions that periodically make forward progress.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

ITE, bit [24]

ITR empty.

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

INTdis, bits [23:22]

When FEAT_RME is implemented:

Interrupt and SError exception disable. Disables taking interrupts and SError exceptions in Non-debug state.

INTdis	Meaning
`0b00`	This bit has no effect on the masking of interrupts and SError exceptions.
`0b01`	If ExternalInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Non-secure state are masked. If ExternalSecureInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Secure state are masked. If ExternalRootInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Root state are masked. If ExternalRealmInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Realm state are masked.

Note

This control affects both physical and virtual interrupts and SError exceptions.

When OSLSR_EL1.OSLK is 1, this field can be indirectly read and written through the following System registers:

- MDSCR_EL1.
- DBGDSCRext.

The Effective value of this field is `0b00` when ExternalInvasiveDebugEnabled() is FALSE.

When FEAT_RME is implemented, bit[23] of this register is RES0.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'00'`.

When FEAT_Debugv8p4 is implemented:

Interrupt and SError exception disable. Disables taking interrupts and SError exceptions in Non-debug state.

INTdis	Meaning
`0b00`	Masking of interrupts and SError exceptions is controlled by PSTATE and interrupt routing controls.
`0b01`	If ExternalInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Non-secure state are masked. If ExternalSecureInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Secure state are masked.

Note

This control affects both physical and virtual interrupts and SError exceptions.

When OSLSR_EL1.OSLK is 1, this field can be indirectly read and written through the following System registers:

- MDSCR_EL1.
- DBGDSCRext.

The Effective value of this field is `0b00` when ExternalInvasiveDebugEnabled() is FALSE.

When FEAT_Debugv8p4 is implemented, bit[23] of this register is RES0.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'00'`.

Otherwise:

Interrupt and SError exception disable. Disables taking interrupts and SError exceptions in Non-debug state.

INTdis	Meaning
`0b00`	Masking of interrupts and SError exceptions is controlled by PSTATE and interrupt routing controls.
`0b01`	If ExternalInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Non-secure EL1 are masked.
`0b10`	If ExternalInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Non-secure state are masked. If ExternalSecureInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Secure EL1 are masked.
`0b11`	If ExternalInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Non-secure state are masked. If ExternalSecureInvasiveDebugEnabled() is TRUE, then all interrupts and SError exceptions taken to Secure state are masked.

Note

This control affects both physical and virtual interrupts and SError exceptions.

When OSLSR_EL1.OSLK is 1, this field can be indirectly read and written through the following System registers:

- MDSCR_EL1.
- DBGDSCRext.

The Effective value of this field is `0b00` when ExternalInvasiveDebugEnabled() is FALSE.

Support for the values `0b01` and `0b10` is IMPLEMENTATION DEFINED. If these values are not supported, they are reserved. If programmed with a reserved value, the PE behaves as if EDSCR.INTdis has been programmed with a defined value, other than for a direct read of EDSCR, and the value returned by a read of EDSCR.INTdis is UNKNOWN.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'00'`.

TDA, bit [21]

Traps accesses to the following debug System registers:

- AArch64: DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGWCR<n>_EL1, DBGWVR<n>_EL1.
- AArch32: DBGBCR<n>, DBGBVR<n>, DBGBXVR<n>, DBGWCR<n>, DBGWVR<n>.

TDA	Meaning
0b0	Accesses to debug System registers do not generate a Software Access Debug event.
0b1	Accesses to debug System registers generate a Software Access Debug event, if OSLSR_EL1.OSLK is 0 and if halting is allowed.

When OSLSR_EL1.OSLK is 1, this bit can be indirectly read and written through MDSCR_EL1 and DBGDSCRext.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

MA, bit [20]

Memory access mode. Controls the use of memory-access mode for accessing ITR and the DCC. This bit is ignored if in Non-debug state and set to zero on entry to Debug state.

Possible values of this field are:

MA	Meaning
0b0	Normal access mode.
0b1	Memory access mode.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

SC2, bit [19]

When FEAT_PCSRv8 is implemented, EL2 is implemented, FEAT_Debugv8p1 is implemented, and FEAT_PCSRv8p2 is not implemented:

Sample CONTEXTIDR_EL2. Controls whether the PC Sample-based Profiling Extension samples CONTEXTIDR_EL2 or VTTBR_EL2.VMID.

SC2	Meaning
0b0	Sample VTTBR_EL2.VMID.
0b1	Sample CONTEXTIDR_EL2.

When OSLSR_EL1.OSLK is 1, this bit can be indirectly read and written through MDSCR_EL1 and DBGDSCRext.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

NS, bit [18]

When FEAT_RME is implemented:

Non-secure status. Together with the NSE field, gives the current Security state:

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

Otherwise:

Non-secure status. In Debug state, gives the current Security state:

NS	Meaning
0b0	Secure state.
0b1	Non-secure state.

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

Bit [17]

Reserved, RES0.

SDD, bit [16]

When FEAT_RME is implemented:

EL3 debug disabled.

On entry to Debug state:

- If entering from EL3, SDD is set to 0.
- Otherwise, SDD is set to the inverse of `ExternalRootInvasiveDebugEnabled()`.

In Debug state, the value of SDD does not change, even if `ExternalRootInvasiveDebugEnabled()` changes.

In Non-debug state, SDD returns the inverse of `ExternalRootInvasiveDebugEnabled()`.

Access to this field is **RO**.

Otherwise:

Secure debug disabled.

On entry to Debug state:

- If entering in Secure state, SDD is set to 0.
- If entering in Non-secure state, SDD is set to the inverse of `ExternalSecureInvasiveDebugEnabled()`.

In Debug state, the value of the SDD bit does not change, even if `ExternalSecureInvasiveDebugEnabled()` changes.

In Non-debug state:

- SDD returns the inverse of `ExternalSecureInvasiveDebugEnabled()`. If the authentication signals that control `ExternalSecureInvasiveDebugEnabled()` change, a context synchronization event is required to guarantee their effect.
- This bit is unaffected by the Security state of the PE.

If EL3 is not implemented and the implementation is Non-secure, this bit is RES1.

Access to this field is **RO**.

NSE, bit [15]

When FEAT_RME is implemented:

Together with the NS field, this field gives the current Security state.

For a description of the values derived by evaluating NS and NSE together, see EDSCR.NS.

In Non-debug state, this bit is UNKNOWN.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HDE, bit [14]

Halting debug enable.

HDE	Meaning
0b0	Halting disabled for Breakpoint, Watchpoint and Halt Instruction debug events.
0b1	Halting enabled for Breakpoint, Watchpoint and Halt Instruction debug events.

When OSLSR_EL1.OSLK is 1, this bit can be indirectly read and written through MDSCR_EL1 and DBGDSCRext.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

RW, bits [13:10]

Exception level Execution state status. In Debug state, each bit gives the current Execution state of each Exception level.

RW	Meaning	Applies when
0b1111	Any of the following: • The PE is in Non-debug state. • The PE is at EL0 using AArch64. • The PE is not at EL0. EL1 is using AArch64. If implemented and enabled in the current Security state, EL2 is using AArch64. If implemented, EL3 is using AArch64.	
0b1110	The PE is in Debug state at EL0. EL0 is using AArch32. EL1 is using AArch64. If implemented and enabled in the current Security state, EL2 is using AArch64. If implemented, EL3 is using AArch64.	When FEAT_AA32 is implemented
0b110x	The PE is in Debug state. EL0 and EL1 are using AArch32. EL2 is enabled in the current Security state and is using AArch64. If implemented, EL3 is using AArch64.	When FEAT_AA32 is implemented and EL2 is implemented
0b10xx	The PE is in Debug state. EL0 and EL1 are using AArch32. EL2 is not implemented, disabled in the current Security state, or using AArch32. EL3 is using AArch64.	When FEAT_AA32 is implemented and EL3 is implemented
0b0xxx	The PE is in Debug state. All Exception levels are using AArch32.	When FEAT_AA32 is implemented

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **RAO/WI**.
- Otherwise, access to this field is **RO**.

EL, bits [9:8]

Exception level. In Debug state, gives the current Exception level of the PE.

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RO**.

A, bit [7]

SError exception pending. In Debug state, indicates whether an SError exception is pending:

- If EL2 is enabled in the current Security state, the PE is executing at EL0 or EL1, HCR_EL2.TGE is 0 and either HCR_EL2.AMO is 1 or FEAT_DoubleFault2 is implemented and the Effective value of HCRX_EL2.TMEA is 1, a virtual SError exception.
- Otherwise, a physical SError exception.

A	Meaning
0b0	No SError exception pending.
0b1	SError exception pending.

A debugger can read EDSCR to check whether an SError exception is pending without having to execute further instructions. A pending SError might indicate data from target memory is corrupted.

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

ERR, bit [6]

Cumulative error flag. This bit is set to 1 following exceptions in Debug state and on any signaled overrun or underrun on the DTR or EDITR.

When OSLSR_EL1.OSLK is 1, this bit can be indirectly read and written through MDSCR_EL1 and DBGDSCRext.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Access to this field is **RO**.

STATUS, bits [5:0]

On entering Debug state, the PE sets this field to indicate the reason for halting.

STATUS	Meaning
0b000001	PE is restarting, exiting Debug state.
0b000010	PE is in Non-debug state.
0b000111	Breakpoint.
0b010011	External debug request.
0b011011	Halting step, normal.
0b011111	Halting step, exclusive.
0b100011	OS Unlock Catch.
0b100111	Reset Catch.
0b101011	Watchpoint.
0b101111	HLT instruction.
0b110011	Software access to debug register.
0b110111	Exception Catch.
0b111011	Halting step, no syndrome.

All other values are reserved.

The PE resets into Non-debug state. However, if the PE enters Debug state immediately after reset, then the reset value is overwritten with the reason for halting.

The reset behavior of this field is:

- On a Cold reset, this field resets to '000010'.

Access to this field is **RO**.

Accessing EDSCR

EDSCR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x088	EDSCR

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDSCR2, External Debug Status and Control Register 2

The EDSCR2 characteristics are:

Purpose

Main control register 2 for the debug implementation.

Configuration

External register EDSCR2 bits [3, 1] are architecturally mapped to AArch64 System register MDSCR_EL1[35, 33].

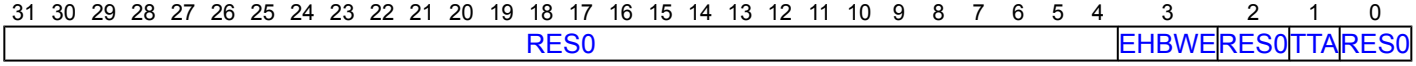
EDSCR2 is in the Core power domain.

This register is present only when FEAT_Debugv8p9 is implemented or FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to EDSCR2 are RES0.

Attributes

EDSCR2 is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

EHBWE, bit [3]

When FEAT_Debugv8p9 is implemented:

Extended Halting Breakpoint and Watchpoint Enable. Enables use of additional breakpoints or watchpoints.

EHBWE	Meaning
0b0	Halting disabled for Breakpoint and Watchpoint debug events generated by each breakpoint <n> and Watchpoint <n>, where n is greater than or equal to 16.
0b1	Breakpoints and Watchpoint debug events are not affected by this mechanism.

When OSLSR_EL1.OSLK is 1, this field can be read and written through the MDSCR_EL1 System register.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [2]

Reserved, RES0.

TTA, bit [1]

When FEAT_TRBE_EXT is implemented or FEAT_ETEv1p3 is implemented:

Trap Trace Accesses.

Traps access to the following System registers:

AArch64: TRBBASER_EL1, TRBIDR_EL1, TRBLIMITR_EL1, TRBMAR_EL1, TRBMPAM_EL1, TRBPTR_EL1, TRBSR_EL1, TRBTRG_EL1, TRCACATR<n>, TRCACVR<n>, TRCAUTHSTATUS, TRCAUXCTLR, TRCBBCTLR, TRCCCCTLR, TRCCIDCCTLR0, TRCCIDCCTLR1, TRCCIDCVR<n>, TRCCLAIMCLR, TRCCLAIMSET, TRCCNTCTLR<n>, TRCCNTRLDVR<n>, TRCCNTVR<n>, TRCCONFIGR, TRCDEVARCH, TRCDEVID, TRCEVENTCTL0R, TRCEVENTCTL1R, TRCEXTINSELR<n>, TRCIDR0, TRCIDR1, TRCIDR2, TRCIDR3, TRCIDR4, TRCIDR5, TRCIDR6, TRCIDR7, TRCIDR8, TRCIDR9, TRCIDR10, TRCIDR11, TRCIDR12, TRCIDR13, TRCIMSPEC0, TRCIMSPEC<n>, TRCITEEDCR, TRCOSLSR, TRCPRGCTLR, TRCQCTLR, TRCRSCTLR<n>, TRCRSR, TRCSEQEVR<n>, TRCSEQRSTEVR, TRCSEQSTR, TRCSSCCR<n>, TRCSSCSR<n>, TRCSSPCICR<n>, TRCSTALLCTLR, TRCSTATR, TRCSYNCPR, TRCTRACEIDR, TRCTSCTLR, TRCVICTLR, TRCVIIECTLR, TRCVIPCSSCTLR, TRCVISSCTLR, TRCVMIDCCTLR0, TRCVMIDCCTLR1, and TRCVMIDCVR<n>.

TTA	Meaning
0b0	Accesses to trace System registers do not generate a Software Access debug event.
0b1	Accesses to trace System registers generate a Software Access debug event, if OSLSR_EL1.OSLK is 0 and if halting is allowed.

When OSLSR_EL1.OSLK is 1, this field can be read and written through the MDSCR_EL1 System register.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [0]

Reserved, RES0.

Accessing EDSCR2

EDSCR2 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x028	EDSCR2

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDVIDSR, External Debug Virtual Context Sample Register

The EDVIDSR characteristics are:

Purpose

Contains sampled values captured on reading EDPCSR[31:0].

Configuration

EDVIDSR is in the Core power domain.

This register is present only when FEAT_PCSRv8 is implemented and FEAT_PCSRv8p2 is not implemented. Otherwise, direct accesses to EDVIDSR are RES0.

If FEAT_Debugv8p1 is implemented, the format of this register differs depending on the value of EDSCR.SC2.

Implemented only if the OPTIONAL PC Sample-based Profiling Extension is implemented in the external debug registers space.

When the PC Sample-based Profiling Extension is implemented in the external debug registers space, if EL2 is not implemented and EL3 is not implemented, it is IMPLEMENTATION DEFINED whether EDVIDSR is implemented.

Note

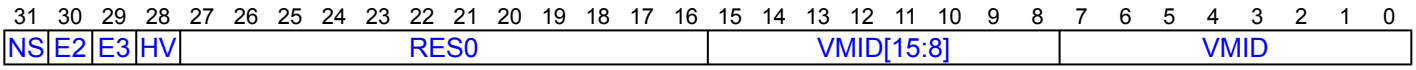
FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors registers space.

Attributes

EDVIDSR is a 32-bit register.

Field descriptions

When FEAT_Debugv8p1 is not implemented or EDSCR.SC2 == 0:



This format applies in all Armv8.0 implementations.

NS, bit [31]

Non-secure state sample. Indicates the Security state associated with the most recent EDPCSR sample.

If EL3 is not implemented, this bit indicates the Effective value of SCR.NS.

NS	Meaning
0b0	Sample is from Secure state.
0b1	Sample is from Non-secure state.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E2, bit [30]

When EL2 is implemented:

Exception level 2 status sample. Indicates whether the most recent EDPCSR sample was associated with EL2.

E2	Meaning
0b0	Sample is not from EL2.
0b1	Sample is from EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E3, bit [29]

When EL3 is implemented and FEAT_AA64 is implemented:

Exception level 3 status sample. Indicates whether the most recent EDPCSR sample was associated with EL3 using AArch64.

E3	Meaning
0b0	Sample is not from EL3 using AArch64.
0b1	Sample is from EL3 using AArch64.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HV, bit [28]

EDPCSRhi (EDPCSR[63:32]) valid. Indicates whether bits [63:32] of the most recent EDPCSR sample might be nonzero:

HV	Meaning
0b0	Bits[63:32] of the most recent EDPCSR sample are zero.
0b1	Bits[63:32] of the most recent EDPCSR sample might be nonzero.

An EDVIDSR.HV value of 1 does not mean that the value of EDPCSRhi is nonzero. An EDVIDSR.HV value of 0 is a hint that EDPCSRhi (EDPCSR[63:32]) does not need to be read.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [27:16]

Reserved, RES0.

VMID[15:8], bits [15:8]

When FEAT_VMID16 is implemented and EL2 is implemented:

Extension to VMID[7:0]. For more information, see VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID, bits [7:0]

When EL2 is implemented:

VMID sample. The VMID associated with the most recent EDPCSRlo (EDPCSR[31:0]) sample. When the most recent EDPCSR sample was generated:

- This field is RES0 if any of the following apply:
  - The PE is executing in Secure state.
  - The PE is executing at EL2.
- Otherwise:
  - If EL2 is using AArch64 and either FEAT_VMID16 is not implemented or VTCR_EL2.VS is 1, this field is set to VTTBR_EL2.VMID.
  - If EL2 is using AArch64, FEAT_VMID16 is implemented, and VTCR_EL2.VS is 0, PMVIDSR.VMID[7:0] is set to VTTBR_EL2.VMID[7:0] and PMVIDSR.VMID[15:8] is RES0.
  - If EL2 is using AArch32, this field is set to VTTBR.VMID.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When EL2 is implemented, FEAT_Debugv8p1 is implemented, and EDSCR.SC2 == 1:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CONTEXTIDR_EL2																															

CONTEXTIDR_EL2, bits [31:0]

Context ID. The value of CONTEXTIDR_EL2 that is associated with the most recent EDPCSR sample. When the most recent EDPCSR sample is generated:

- If the PE is not executing at EL3, EL2 is using AArch64, and EL2 is enabled in the current Security state, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- Otherwise, this field is set to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing EDVIDSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

EDVIDSR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x0A8	EDVIDSR

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDWAR, External Debug Watchpoint Address Register

The EDWAR characteristics are:

Purpose

Returns the virtual data address being accessed when a Watchpoint Debug Event was triggered.

Configuration

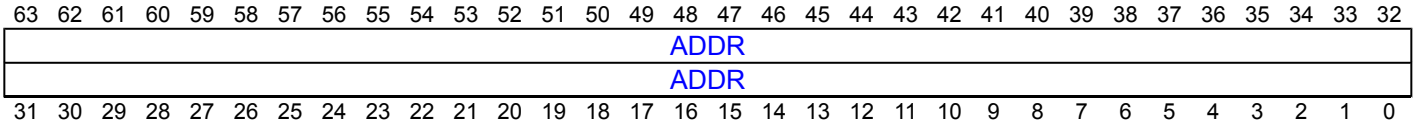
EDWAR is in the Core power domain.

The value of this register is UNKNOWN if the PE is in Non-debug state, or if EDSCR.STATUS is not `0b101011`.

Attributes

EDWAR is a 64-bit register.

Field descriptions



ADDR, bits [63:0]

Watchpoint address. The data virtual address being accessed when a Watchpoint Debug Event was triggered and caused entry to Debug state. This address must be within a naturally-aligned block of memory of power-of-two size no larger than the DC ZVA block size.

The value of this register is UNKNOWN if the PE is in Non-debug state, or if Debug state was entered other than for a Watchpoint debug event.

The value of EDWAR[63:32] is UNKNOWN if Debug state was entered for a Watchpoint debug event taken from AArch32 state.

The EDWAR is subject to the same alignment rules as the reporting of a watchpointed address in the FAR. See 'Exception syndrome information and preferred return address'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing EDWAR

EDWAR can be accessed through the external debug interface:

Component	Offset	Instance
Debug	`0x030`	EDWAR

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRACR, Access Configuration Register

The ERRACR characteristics are:

Purpose

Controls visibility of error records.

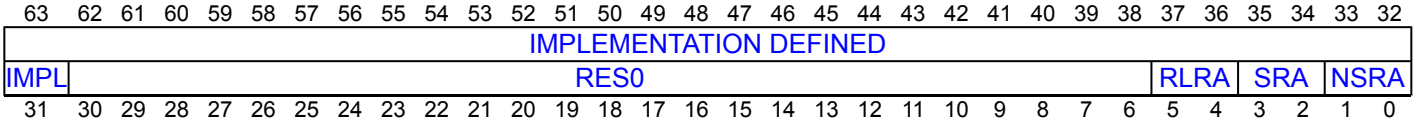
Configuration

This register is present only when (Root state is implemented or Secure state is implemented) and an implementation implements ERRACR. Otherwise, direct accesses to ERRACR are RES0.

Attributes

ERRACR is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:32]

IMPLEMENTATION DEFINED observation controls. Additional IMPLEMENTATION DEFINED access control bits.

IMPL, bit [31]

IMPL	Meaning
0b1	Indicates ERRACR is present.

Access to this field is **RAO/WI**.

Bits [30:6]

Reserved, RES0.

RLRA, bits [5:4]

When FEAT_RME is implemented and the error record group allows configuration of Secure and Realm register accesses:

Realm Restricted Access. Controls Realm access to error records and interrupt configuration registers in the error record group.

RLRA	Meaning
0b00	Realm access is disabled. All error record, ERR<irq>CR<m>, and ERRIRQSR registers are RAZ/WI to Realm accesses.
0b01	Realm read access is enabled. Realm writes are ignored.
0b11	Realm read/write access is allowed. If the error record group supports MSIs, generated MSIs are always Non-secure.

All other values are reserved.

This control applies to all error record registers (ERR<n>*, including fault injection registers ERR<n>PFG* if implemented), and interrupt configuration registers (ERR<irq>CR<m> and ERRIRQSR, if implemented) in the error record group. The effect on any IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

When Realm access to error records is disabled, a Realm read of ERRGSR will return the error record status for the error records that cannot be accessed.

When Realm access is fully or partially disabled, the effect on Realm accesses to IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

Note

Realm access to error records is enabled from reset.

The reset domain and value of this field is IMPLEMENTATION DEFINED, and depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.
- On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RAZ/WI.

SRA, bits [3:2]

When Secure state is implemented, FEAT_RME is implemented, and the error record group allows configuration of Secure and Realm register accesses:

Secure Restricted Access. Controls Secure access to error records and interrupt configuration registers in the error record group.

SRA	Meaning
0b00	Secure access is disabled. All error record, ERR<irq>CR<m>, and ERRIRQSR registers are RAZ/WI to Secure accesses.
0b01	Secure read access is enabled. Secure writes are ignored.
0b11	Secure read/write access is allowed.

All other values are reserved.

This control applies to all error record registers (ERR<n>*, including fault injection registers ERR<n>PFG* if implemented), and interrupt configuration registers (ERR<irq>CR<m> and ERRIRQSR, if implemented) in the error record group. The effect on any IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

When Secure access to error records is disabled, a Secure read of ERRGSR will return the error record status for the error records that cannot be accessed.

When Secure access is fully or partially disabled, the effect on Secure accesses to IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

Note

Secure access to error records is enabled from reset.

The reset domain and value of this field is IMPLEMENTATION DEFINED, and depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.
- On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RAZ/WI.

NSRA, bits [1:0]

Non-secure Restricted Access. Controls Non-secure access to error records and interrupt configuration registers in the error record group.

NSRA	Meaning
0b00	Non-secure access is disabled. All error record, ERR<irq>CR<m>, and ERRIRQSR registers are RAZ/WI to Non-secure accesses.
0b01	Non-secure read access is enabled. Non-secure writes are ignored.
0b11	Non-secure read/write access is allowed. If the error record group supports MSIs, generated MSIs are always Non-secure.

All other values are reserved.

This control applies to all error record registers (ERR<n>*, including fault injection registers ERR<n>PFG* if implemented), and interrupt configuration registers (ERR<irq>CR<m> and ERRIRQSR, if implemented) in the error record group. The effect on any IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

When Non-secure access to error records is disabled, a Non-secure read of ERRGSR will return the error record status for the error records that cannot be accessed.

When Non-secure access is fully or partially disabled, the effect on Non-secure accesses to IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

Note

Non-secure access to error records is enabled from reset.

If FEAT_RME is implemented and ERRACR.{RLRA, SRA} are not implemented, then ERRACR.NSRA applies to all Security states other than Root.

The reset domain and value of this field is IMPLEMENTATION DEFINED, and depends on the security policy of the component implementing this register.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.
- On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing ERRACR

This section shows the offset of ERRACR when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRACR.

ERRACR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xE40	ERRACR

Accessible as follows:

- When (FEAT_RME is implemented and an access is not Root) or an access is Non-secure, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCIDR0, Component Identification Register 0

The ERRCIDR0 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

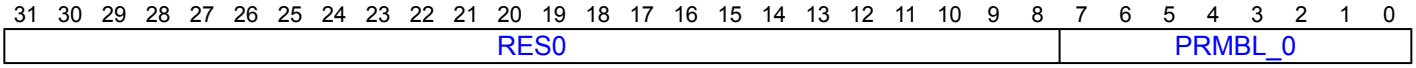
Implementation of this register is OPTIONAL.

ERRCIDR0 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Component identification preamble, segment 0.

Reads as `0x0D`.

Access to this field is **RO**.

Accessing ERRCIDR0

This section shows the offset of ERRCIDR0 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRCIDR0.

ERRCIDR0 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0xFF0`	ERRCIDR0

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCIDR1, Component Identification Register 1

The ERRCIDR1 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

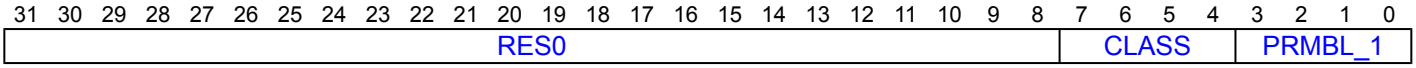
Implementation of this register is OPTIONAL.

ERRCIDR1 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

CLASS	Meaning
0b1111	Generic peripheral with IMPLEMENTATION DEFINED register layout.

Other values are defined by the CoreSight Architecture.

This field reads as 0xF.

Access to this field is **RO**.

PRMBL_1, bits [3:0]

Component identification preamble, segment 1.

Reads as 0b0000.

Access to this field is **RO**.

Accessing ERRCIDR1

This section shows the offset of ERRCIDR1 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRCIDR1.

ERRCIDR1 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xFF4	ERRCIDR1

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCIDR2, Component Identification Register 2

The ERRCIDR2 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

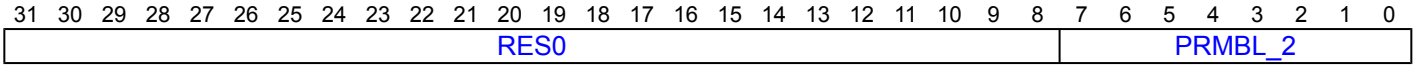
Implementation of this register is OPTIONAL.

ERRCIDR2 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCIDR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Component identification preamble, segment 2.

Reads as `0x05`.

Access to this field is **RO**.

Accessing ERRCIDR2

This section shows the offset of ERRCIDR2 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRCIDR2.

ERRCIDR2 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0xFF8`	ERRCIDR2

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCIDR3, Component Identification Register 3

The ERRCIDR3 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

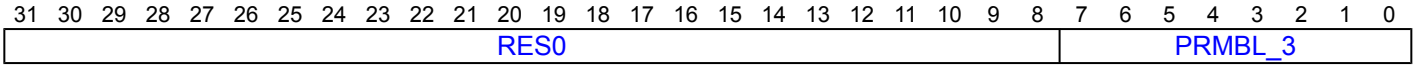
Implementation of this register is OPTIONAL.

ERRCIDR3 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCIDR3 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Component identification preamble, segment 3.

Reads as `0xB1`.

Access to this field is **RO**.

Accessing ERRCIDR3

This section shows the offset of ERRCIDR3 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRCIDR3.

ERRCIDR3 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0xFFC`	ERRCIDR3

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCRICR0, Critical Error Interrupt Configuration Register 0

The ERRCRICR0 characteristics are:

Purpose

Critical Error Interrupt configuration register.

Configuration

This register is present only when (the Critical Error Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRCRICR0 are RES0.

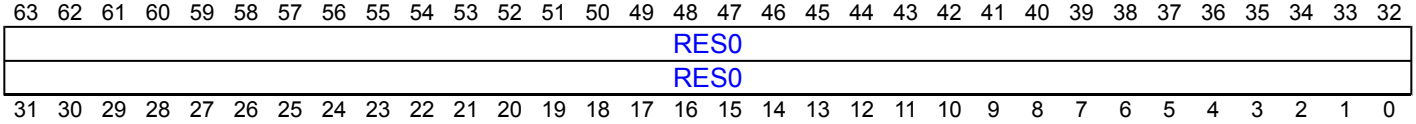
ERRCRICR0 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCRICR0 is a 64-bit register.

Field descriptions

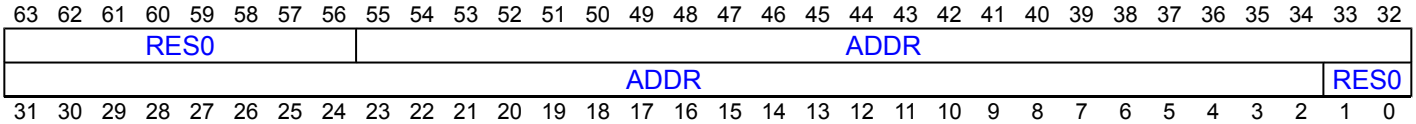
When the Critical Error Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [63:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Critical Error Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [63:56]

Reserved, RES0.

ADDR, bits [55:2]

Message Signaled Interrupt address. (ERRCRICR0.ADDR << 2) is the address that the component writes to when signaling the Critical Error Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the component is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

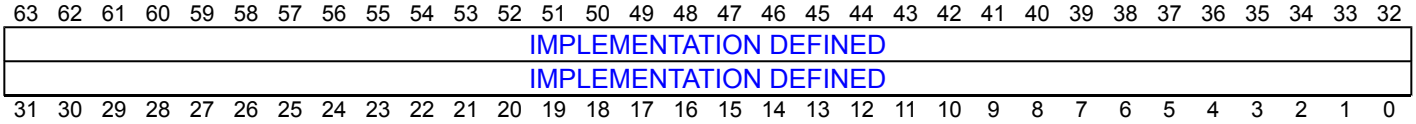
The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR registers:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRCRICR0

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRCRICR0 are IMPLEMENTATION DEFINED .

This section shows the offset of ERRCRICR0 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRCRICR0.

ERRCRICR0 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xEA0	ERRCRICR0

Accessible as follows:

- When the implementation uses message-signaled interrupts, (an access is Non-secure or an access is Realm), the implementation uses the recommended layout for the ERRIRQCR registers, and ERRCRICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCRICR1, Critical Error Interrupt Configuration Register 1

The ERRCRICR1 characteristics are:

Purpose

Critical Error Interrupt configuration register.

Configuration

This register is present only when (the Critical Error Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRCRICR1 are RES0.

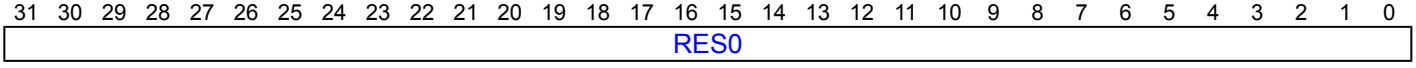
ERRCRICR1 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCRICR1 is a 32-bit register.

Field descriptions

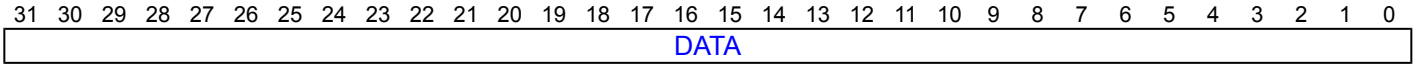
When the Critical Error Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Critical Error Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



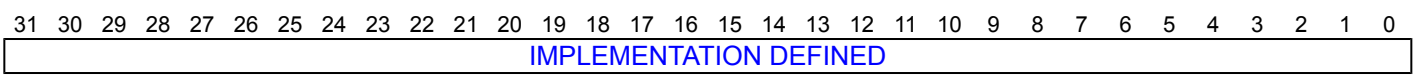
DATA, bits [31:0]

Payload for the message signaled interrupt.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the implementation does not use the recommended layout for the ERRIRQCR registers:



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRCRICR1

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRCRICR1 are IMPLEMENTATION DEFINED .

This section shows the offset of ERRCRICR1 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRCRICR1.

ERRCRICR1 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xEA8	ERRCRICR1

Accessible as follows:

- When the implementation uses message-signaled interrupts, (an access is Non-secure or an access is Realm), the implementation uses the recommended layout for the ERRIRQCR registers, and ERRCRICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCRICR2, Critical Error Interrupt Configuration Register 2

The ERRCRICR2 characteristics are:

Purpose

Critical Error Interrupt control and configuration register.

Configuration

This register is present only when (the Critical Error Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRCRICR2 are RES0.

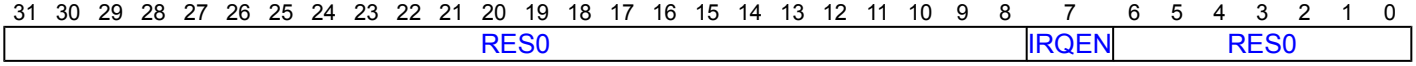
ERRCRICR2 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCRICR2 is a 32-bit register.

Field descriptions

When the Critical Error Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

Interrupts enable. Enables generation of interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

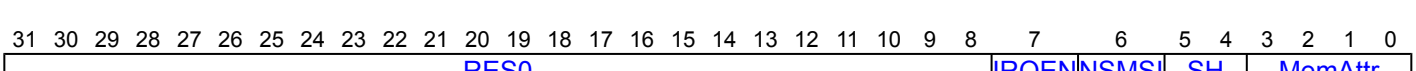
The reset behavior of this field is:

- On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [6:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Critical Error Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

When the component supports disabling message signaled interrupts:

Message signaled interrupt enable. Enables generation of message signaled interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Message signaled interrupts are always enabled.

NSMSI, bit [6]

When the component supports configuring the physical address space for message signaled interrupts:

Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.

NSMSI	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When an access is Non-secure, access to this field is **RO**.
- When an access is Realm, access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Non-secure message signaled interrupt.

The physical address space for message signaled interrupts is IMPLEMENTATION DEFINED.

SH, bits [5:4]

When the component supports configuring the Shareability domain for message signaled interrupts:

Shareability. Defines the Shareability domain for message signaled interrupts.

SH	Meaning
0b00	Not shared.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when ERRCRICR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Shareability.

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

MemAttr, bits [3:0]

When the component supports configuring the memory type for message signaled interrupts:

Memory type. Defines the memory type and attributes for message signaled interrupts.

MemAttr	Meaning
0b0000	Device-nGnRnE memory.
0b0001	Device-nGnRE memory.
0b0010	Device-nGRE memory.
0b0011	Device-GRE memory.
0b0101	Normal memory, Inner Non-cacheable, Outer Non-cacheable.
0b0110	Normal memory, Inner Write-Through, Outer Non-cacheable.
0b0111	Normal memory, Inner Write-Back, Outer Non-cacheable.
0b1001	Normal memory, Inner Non-cacheable, Outer Write-Through.
0b1010	Normal memory, Inner Write-Through, Outer Write-Through.
0b1011	Normal memory, Inner Write-Back, Outer Write-Through.
0b1101	Normal memory, Inner Non-cacheable, Outer Write-Back.
0b1110	Normal memory, Inner Write-Through, Outer Write-Back.
0b1111	Normal memory, Inner Write-Back, Outer Write-Back.

All other values are reserved.

Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Memory type.

The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

When the implementation does not use the recommended layout for the ERRIRQCR registers:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMPLEMENTATION DEFINED																															

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRCRICR2

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRCRICR2 are IMPLEMENTATION DEFINED .

This section shows the offset of ERRCRICR2 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRCRICR2.

ERRCRICR2 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xEAC	ERRCRICR2

Accessible as follows:

- When the implementation uses message-signaled interrupts, (an access is Non-secure or an access is Realm), the implementation uses the recommended layout for the ERRIRQCR registers, and ERRCRICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRDEVAFF, Device Affinity Register

The ERRDEVAFF characteristics are:

Purpose

For a group of error records that has affinity with a single PE or a group of PEs, ERRDEVAFF is a copy of MPIDR_EL1 or part of MPIDR_EL1:

- If the group of error records has affinity with a single PE, the affinity level is 0, then ERRDEVAFF reads the same value as MPIDR_EL1, and ERRDEVAFF.F0V reads-as-one to indicate affinity level 0.
- If the group of error records has affinity with a group of PEs, the affinity level is 1, 2, or 3, then parts of ERRDEVAFF reads the same value as parts of MPIDR_EL1, and the rest of ERRDEVAFF indicates the level.

For example, if the group of PEs is a subset of the PEs at affinity level 1 then all of the following are true:

- All the PEs in the group have the same values in MPIDR_EL1.{Aff3,Aff2}, and these values are equal to ERRDEVAFF.{Aff3,Aff2}.
- ERRDEVAFF.Aff1 is nonzero and not `0x80`, and ERRDEVAFF.{Aff0,F0V} read-as-zero, to indicate at least affinity level 1. The subset of PEs at level 1 that the group of error records has affinity with is indicated by the least-significant set bit in ERRDEVAFF.Aff1. In this example, if ERRDEVAFF.Aff1[2:0] is `0b100`, then the group of error records has affinity with the up-to 8 PEs that have MPIDR_EL1.Aff1[7:3] == ERRDEVAFF.Aff1[7:3].

Depending on the IMPLEMENTATION DEFINED nature of the system, it might be possible that ERRDEVAFF is read before system firmware has configured the group of error records or the PE or group of PEs that the group of error records has affinity with. When this is the case, ERRDEVAFF reads as zero.

If RAS System Architecture v1.1 is not implemented then ERRDEVAFF can only describe a group of error records that is affine with a single PE or all the PEs at an affinity level.

Configuration

This register is present only when the group of error records has affinity with a PE or cluster of PEs and an implementation implements ERRDEVAFF. Otherwise, direct accesses to ERRDEVAFF are RES0.

ERRDEVAFF is implemented only as part of a memory-mapped group of error records.

Arm deprecates use of this register by software

Attributes

ERRDEVAFF is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																								Aff3							
F0V	U	RES0					MT	Aff2												Aff1								Aff0			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

PE affinity level 3. The MPIDR_EL1.Aff3 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

F0V, bit [31]

Indicates that the ERRDEVAFF.Aff0 field is valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F0V	Meaning
0b0	ERRDEVAFF.Aff0 is not valid, and the PE affinity is above level 0 or a subset of level 0.
0b1	ERRDEVAFF.Aff0 is valid, and the PE affinity is at level 0.

Access to this field is **RO**.

U, bit [30]

When ERRDEVAFF.F0V == 1:

Uniprocessor. The MPIDR_EL1.U field, viewed from the highest Exception level of the associated PE.

Otherwise:

Reserved, UNKNOWN.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

When ERRDEVAFF.F0V == 1:

Multithreaded. The MPIDR_EL1.MT field, viewed from the highest Exception level of the associated PE.

Otherwise:

Reserved, UNKNOWN.

Aff2, bits [23:16]

When !IsZero(ERRDEVAFF.[Aff1,Aff0,F0V]):

PE affinity level 2. The MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

PE affinity level 2. Defines part of the MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff2	Meaning
0bxxxxxxx1	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:1] is the value of MPIDR_EL1.Aff2[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:2] is the value of MPIDR_EL1.Aff2[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:3] is the value of MPIDR_EL1.Aff2[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:4] is the value of MPIDR_EL1.Aff2[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:5] is the value of MPIDR_EL1.Aff2[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7:6] is the value of MPIDR_EL1.Aff2[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	PE affinity is the subset of level 2 where ERRDEVAFF.Aff2[7] is the value of MPIDR_EL1.Aff2[7], viewed from the highest Exception level of the associated PEs.
0x80	PE affinity is at level 3.

Access to this field is **RO**.

Aff1, bits [15:8]

When !IsZero(ERRDEVAFF.[Aff0,F0V]):

PE affinity level 1. The MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

PE affinity level 1. Defines part of the MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff1	Meaning
0x00	PE affinity is above level 2 or a subset of level 2.
0bxxxxxxx1	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:1] is the value of MPIDR_EL1.Aff1[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:2] is the value of MPIDR_EL1.Aff1[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:3] is the value of MPIDR_EL1.Aff1[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:4] is the value of MPIDR_EL1.Aff1[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:5] is the value of MPIDR_EL1.Aff1[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7:6] is the value of MPIDR_EL1.Aff1[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	PE affinity is the subset of level 1 where ERRDEVAFF.Aff1[7] is the value of MPIDR_EL1.Aff1[7], viewed from the highest Exception level of the associated PEs.
0x80	PE affinity is at level 2.

Access to this field is **RO**.

Aff0, bits [7:0]

When ERRDEVAFF.F0V == 1:

PE affinity level 0. The MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

PE affinity level 0. Defines part of the MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff0	Meaning
0x00	PE affinity is above level 1 or a subset of level 1.
0bxxxxxxx1	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:1] is the value of MPIDR_EL1.Aff0[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:2] is the value of MPIDR_EL1.Aff0[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:3] is the value of MPIDR_EL1.Aff0[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:4] is the value of MPIDR_EL1.Aff0[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:5] is the value of MPIDR_EL1.Aff0[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7:6] is the value of MPIDR_EL1.Aff0[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	PE affinity is the subset of level 0 where ERRDEVAFF.Aff0[7] is the value of MPIDR_EL1.Aff0[7], viewed from the highest Exception level of the associated PEs.
0x80	PE affinity is at level 1.

Access to this field is **RO**.

Accessing ERRDEVAFF

This section shows the offset of ERRDEVAFF when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRDEVAFF.

ERRDEVAFF can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xFA8	ERRDEVAFF

Accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRDEVARCH, Device Architecture Register

The ERRDEVARCH characteristics are:

Purpose

Provides discovery information for the component.

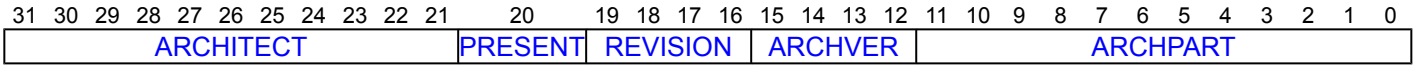
Configuration

ERRDEVARCH is implemented only as part of a memory-mapped group of error records.

Attributes

ERRDEVARCH is a 32-bit register.

Field descriptions



ARCHITECT, bits [31:21]

Defines the architect of the component. For RAS, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, `0b0100`.

Bits [27:21] are the JEP106 identification code, `0b0111011`.

Reads as `0b01000111011`.

Access to this field is **RO**.

PRESENT, bit [20]

DEVARCH present. Indicates that the ERRDEVARCH register is present.

Reads as `0b1`.

Access to this field is **RO**.

REVISION, bits [19:16]

When UInt(ERRDEVARCH.ARCHPART) == 0xA00 and ERRDEVARCH.ARCHVER == 0b0000:

Revision. Defines the architecture revision of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

REVISION	Meaning
`0b0000`	RAS System Architecture, error record group v1.0.
`0b0001`	RAS System Architecture, error record group v1.1. As `0b0000` and also: • Simplifies ERR<n>STATUS. • Adds support for additional ERR<n>MISC<m> registers. • Adds support for the optional RAS Timestamp Extension. • Adds support for the optional Common Fault Injection Model Extension.

All other values are reserved.

Access to this field is **RO**.

When UInt(ERRDEVARCH.ARCHPART) == 0xA00 and ERRDEVARCH.ARCHVER == 0b0001:

Revision. Defines the architecture revision of the component.

REVISION	Meaning
0b0000	RAS System Architecture, error record group v2.0.

All other values are reserved.

Access to this field is **RO**.

When UInt(ERRDEVARCH.ARCHPART) == 0xA08 and ERRDEVARCH.ARCHVER == 0b0000:

Revision. Defines the architecture revision of the component.

REVISION	Meaning
0b0000	RAS System Architecture, fault injection group v1.0.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

ARCHVER, bits [15:12]

When UInt(ERRDEVARCH.ARCHPART) == 0xA00:

Architecture Version. Defines the architecture version of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ARCHVER	Meaning
0b0000	RAS System Architecture, error record group v1.
0b0001	RAS System Architecture, error record group v2. As 0b0000 and also: • Adds an optional access control register, ERRACR. • Adds an optional control for disabling error counters. • Adds optional fault handling interrupt controls for Deferred errors. • Adds support for continuation and proxy error records. • Adds support for implementing Common Fault Injection Mechanism registers in a separate page from the error record registers. • Adds support for simple interrupt control registers. • Defines fields in ERRDEVID that describe these properties.

All other values are reserved.

ERRDEVARCH.ARCHVER and ERRDEVARCH.ARCHPART are also defined as a single field, ERRDEVARCH.ARCHID, so that ERRDEVARCH.ARCHVER is ERRDEVARCH.ARCHID[15:12].

Access to this field is **RO**.

When UInt(ERRDEVARCH.ARCHPART) == 0xA08:

Architecture Version. Defines the architecture version of the component.

ARCHVER	Meaning
0b0000	RAS System Architecture, fault injection group v1.

All other values are reserved.

ERRDEVARCH.ARCHVER and ERRDEVARCH.ARCHPART are also defined as a single field, ERRDEVARCH.ARCHID, so that ERRDEVARCH.ARCHVER is ERRDEVARCH.ARCHID[15:12].

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ARCHPART	Meaning
0xA00	RAS System Architecture, error record group.
0xA08	RAS System Architecture, fault injection group.

ERRDEVARCH.ARCHVER and ERRDEVARCH.ARCHPART are also defined as a single field, ERRDEVARCH.ARCHID, so that ERRDEVARCH.ARCHPART is ERRDEVARCH.ARCHID[11:0].

Access to this field is **RO**.

Accessing ERRDEVARCH

This section shows the offset of ERRDEVARCH when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRDEVARCH.

ERRDEVARCH can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xFBC	ERRDEVARCH

Accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRDEVID, Device Configuration Register

The ERRDEVID characteristics are:

Purpose

Provides discovery information for the component.

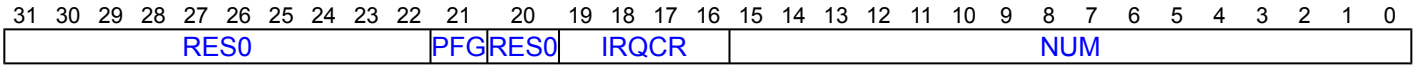
Configuration

ERRDEVID is implemented only as part of a memory-mapped group of error records.

Attributes

ERRDEVID is a 32-bit register.

Field descriptions



Bits [31:22]

Reserved, RES0.

PFG, bit [21]

When RAS System Architecture v2 is implemented:

Common Fault Injection Mechanism. Describes whether any Common Fault Injection Mechanism registers are implemented in the same page as this register.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PFG	Meaning
0b0	Any Common Fault Injection Mechanism registers are implemented in the same page as this register.
0b1	Any Common Fault Injection Mechanism registers are implemented in a separate fault injection group page.

Note

The value of this field does not indicate that any Common Fault Injection Mechanism registers are implemented by the nodes in this error record group. Software must use ERR<n>FR to discover whether each node implements Common Fault Injection Mechanism registers.

Accessing this field has the following behavior:

- When ERRDEVID is part of a fault injection group, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RAZ.

Bit [20]

Reserved, RES0.

IRQCR, bits [19:16]

Interrupt Control registers. Describes whether the interrupt control registers are implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IRQCR	Meaning
0b0000	It is IMPLEMENTATION DEFINED whether any interrupt control registers are implemented.
0b0001	An IMPLEMENTATION DEFINED form of interrupt control registers are implemented.
0b0010	The recommended layout form of interrupt control registers are implemented, for simple interrupts.
0b0011	The recommended layout form of interrupt control registers are implemented, for message-signaled interrupts.
0b1111	Interrupt control registers are not implemented.

All other values are reserved.

Accessing this field has the following behavior:

- When ERRDEVID is part of a fault injection group, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RO**.

NUM, bits [15:0]

Highest numbered index of the error records in this group, plus one. Each implemented record is owned by a node. A node might own multiple records.

This manual describes a group of error records accessed via a standard 4KB memory-mapped peripheral. For a 4KB peripheral, up to 24 error records can be accessed if the Common Fault Injection Model is implemented, and up to 56 otherwise.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRDEVID

This section shows the offset of ERRDEVID when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRDEVID.

ERRDEVID can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xFC8	ERRDEVID

Accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRERICR0, Error Recovery Interrupt Configuration Register 0

The ERRERICR0 characteristics are:

Purpose

Error Recovery Interrupt configuration register.

Configuration

This register is present only when (the Error Recovery Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRERICR0 are RES0.

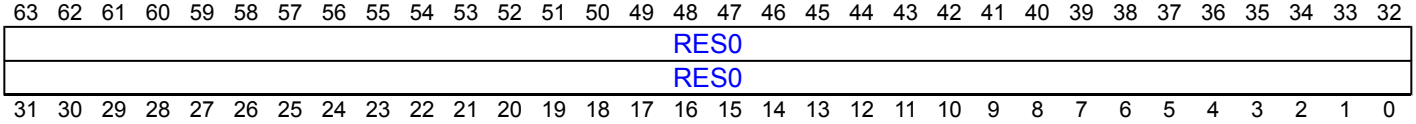
ERRERICR0 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRERICR0 is a 64-bit register.

Field descriptions

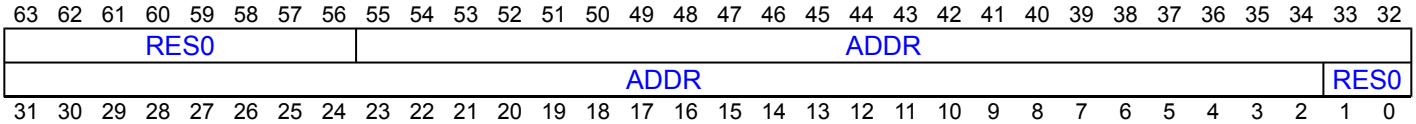
When the Error Recovery Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [63:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Error Recovery Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [63:56]

Reserved, RES0.

ADDR, bits [55:2]

Message Signaled Interrupt address. (ERRERICR0.ADDR << 2) is the address that the component writes to when signaling the Error Recovery Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the component is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

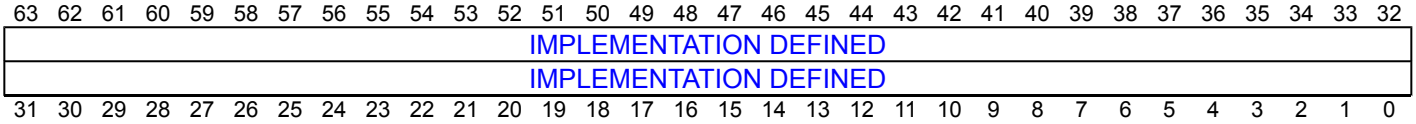
The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR registers:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRERICR0

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRERICR0 are IMPLEMENTATION DEFINED .

This section shows the offset of ERRERICR0 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRERICR0.

ERRERICR0 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xE90	ERRERICR0

Accessible as follows:

- When the implementation uses message-signaled interrupts, (an access is Non-secure or an access is Realm), the implementation uses the recommended layout for the ERRIRQCR registers, and ERRERICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRERICR1, Error Recovery Interrupt Configuration Register 1

The ERRERICR1 characteristics are:

Purpose

Error Recovery Interrupt configuration register.

Configuration

This register is present only when (the Error Recovery Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRERICR1 are RES0.

ERRERICR1 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRERICR1 is a 32-bit register.

Field descriptions

When the Error Recovery Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0																															

Bits [31:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Error Recovery Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA																															

DATA, bits [31:0]

Payload for the message signaled interrupt.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the implementation does not use the recommended layout for the ERRIRQCR registers:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMPLEMENTATION DEFINED																															

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRERICR1

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRERICR1 are IMPLEMENTATION DEFINED .

This section shows the offset of ERRERICR1 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRERICR1.

ERRERICR1 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xE98	ERRERICR1

Accessible as follows:

- When the implementation uses message-signaled interrupts, (an access is Non-secure or an access is Realm), the implementation uses the recommended layout for the ERRIRQCR registers, and ERRERICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRERICR2, Error Recovery Interrupt Configuration Register 2

The ERRERICR2 characteristics are:

Purpose

Error Recovery Interrupt control and configuration register.

Configuration

This register is present only when (the Error Recovery Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRERICR2 are RES0.

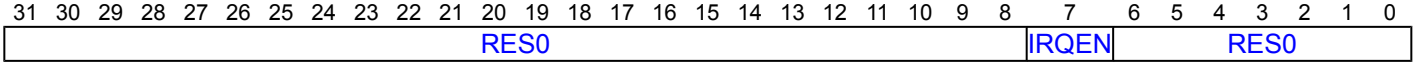
ERRERICR2 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRERICR2 is a 32-bit register.

Field descriptions

When the Error Recovery Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

Interrupts enable. Enables generation of interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

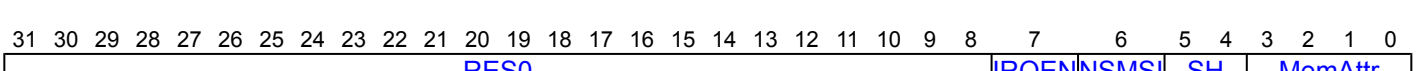
The reset behavior of this field is:

- On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [6:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Error Recovery Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

When the component supports disabling message signaled interrupts:

Message signaled interrupt enable. Enables generation of message signaled interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Message signaled interrupts are always enabled.

NSMSI, bit [6]

When the component supports configuring the physical address space for message signaled interrupts:

Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.

NSMSI	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When an access is Non-secure, access to this field is **RO**.
- When an access is Realm, access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Non-secure message signaled interrupt.

The physical address space for message signaled interrupts is IMPLEMENTATION DEFINED.

SH, bits [5:4]

When the component supports configuring the Shareability domain for message signaled interrupts:

Shareability. Defines the Shareability domain for message signaled interrupts.

SH	Meaning
0b00	Not shared.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when ERRERICR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Shareability.

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

MemAttr, bits [3:0]

When the component supports configuring the memory type for message signaled interrupts:

Memory type. Defines the memory type and attributes for message signaled interrupts.

MemAttr	Meaning
0b0000	Device-nGnRnE memory.
0b0001	Device-nGnRE memory.
0b0010	Device-nGRE memory.
0b0011	Device-GRE memory.
0b0101	Normal memory, Inner Non-cacheable, Outer Non-cacheable.
0b0110	Normal memory, Inner Write-Through, Outer Non-cacheable.
0b0111	Normal memory, Inner Write-Back, Outer Non-cacheable.
0b1001	Normal memory, Inner Non-cacheable, Outer Write-Through.
0b1010	Normal memory, Inner Write-Through, Outer Write-Through.
0b1011	Normal memory, Inner Write-Back, Outer Write-Through.
0b1101	Normal memory, Inner Non-cacheable, Outer Write-Back.
0b1110	Normal memory, Inner Write-Through, Outer Write-Back.
0b1111	Normal memory, Inner Write-Back, Outer Write-Back.

All other values are reserved.

Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Memory type.

The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

When the implementation does not use the recommended layout for the ERRIRQCR registers:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMPLEMENTATION DEFINED																															

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRERICR2

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRERICR2 are IMPLEMENTATION DEFINED .

This section shows the offset of ERRERICR2 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRERICR2.

ERRERICR2 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xE9C	ERRERICR2

Accessible as follows:

- When the implementation uses message-signaled interrupts, (an access is Non-secure or an access is Realm), the implementation uses the recommended layout for the ERRIRQCR registers, and ERRERICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRFHICR0, Fault Handling Interrupt Configuration Register 0

The ERRFHICR0 characteristics are:

Purpose

Fault Handling Interrupt configuration register.

Configuration

This register is present only when (the Fault Handling Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRFHICR0 are RES0.

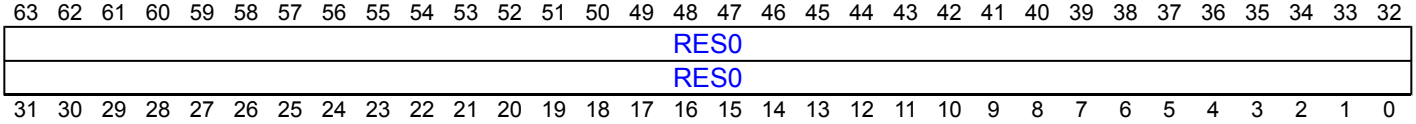
ERRFHICR0 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRFHICR0 is a 64-bit register.

Field descriptions

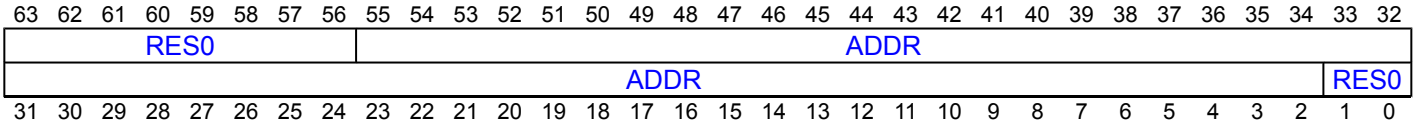
When the Fault Handling Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [63:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Fault Handling Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [63:56]

Reserved, RES0.

ADDR, bits [55:2]

Message Signaled Interrupt address. (ERRFHICR0.ADDR << 2) is the address that the component writes to when signaling the Fault Handling Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the component is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

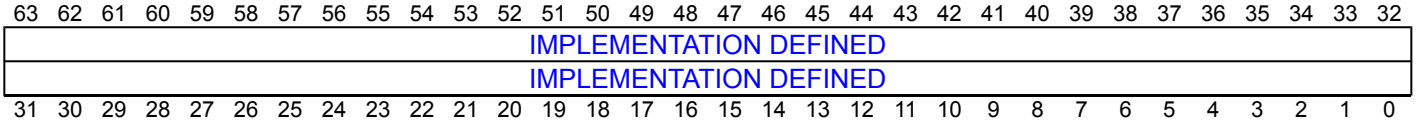
The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR registers:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRFHICR0

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRFHICR0 are IMPLEMENTATION DEFINED .

This section shows the offset of ERRFHICR0 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRFHICR0.

ERRFHICR0 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xE80	ERRFHICR0

Accessible as follows:

- When the implementation uses message-signaled interrupts, (an access is Non-secure or an access is Realm), the implementation uses the recommended layout for the ERRIRQCR registers, and ERRFHICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRFHICR1, Fault Handling Interrupt Configuration Register 1

The ERRFHICR1 characteristics are:

Purpose

Fault Handling Interrupt configuration register.

Configuration

This register is present only when (the Fault Handling Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRFHICR1 are RES0.

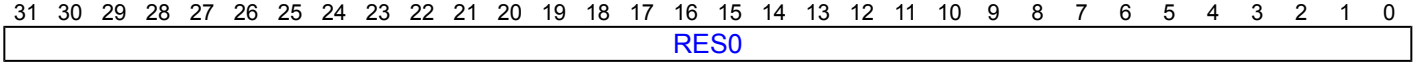
ERRFHICR1 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRFHICR1 is a 32-bit register.

Field descriptions

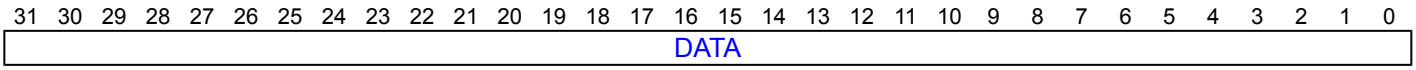
When the Fault Handling Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Fault Handling Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



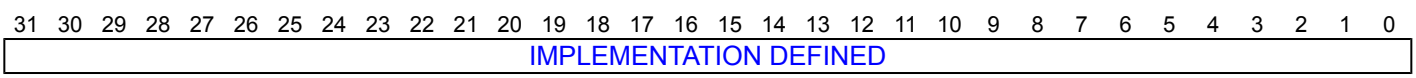
DATA, bits [31:0]

Payload for the message signaled interrupt.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the implementation does not use the recommended layout for the ERRIRQCR registers:



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRFHICR1

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRFHICR1 are IMPLEMENTATION DEFINED .

This section shows the offset of ERRFHICR1 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRFHICR1.

ERRFHICR1 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xE88	ERRFHICR1

Accessible as follows:

- When the implementation uses message-signaled interrupts, (an access is Non-secure or an access is Realm), the implementation uses the recommended layout for the ERRIRQCR registers, and ERRFHICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRFHICR2, Fault Handling Interrupt Configuration Register 2

The ERRFHICR2 characteristics are:

Purpose

Fault Handling Interrupt control and configuration register.

Configuration

This register is present only when (the Fault Handling Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRFHICR2 are RES0.

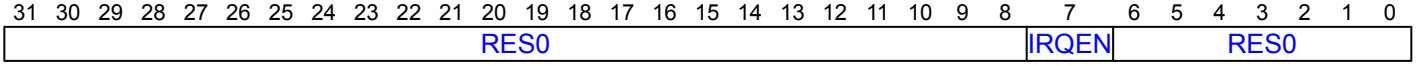
ERRFHICR2 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRFHICR2 is a 32-bit register.

Field descriptions

When the Fault Handling Interrupt is implemented, the implementation uses the recommended layout for the ERRIRQCR registers, and the implementation uses simple interrupts:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

Interrupts enable. Enables generation of interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

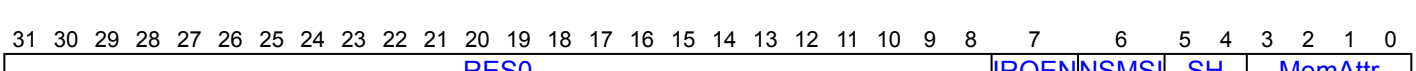
The reset behavior of this field is:

- On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [6:0]

Reserved, RES0.

When the implementation uses message-signaled interrupts, the Fault Handling Interrupt is implemented, and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

When the component supports disabling message signaled interrupts:

Message signaled interrupt enable. Enables generation of message signaled interrupts.

IRQEN	Meaning
0b0	Disabled.
0b1	Enabled.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Message signaled interrupts are always enabled.

NSMSI, bit [6]

When the component supports configuring the physical address space for message signaled interrupts:

Non-secure message signaled interrupt. Defines the physical address space for message signaled interrupts.

NSMSI	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing this field has the following behavior:

- When an access is Non-secure, access to this field is **RO**.
- When an access is Realm, access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Non-secure message signaled interrupt.

The physical address space for message signaled interrupts is IMPLEMENTATION DEFINED.

SH, bits [5:4]

When the component supports configuring the Shareability domain for message signaled interrupts:

Shareability. Defines the Shareability domain for message signaled interrupts.

SH	Meaning
0b00	Not shared.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when ERRFHICR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Shareability.

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

MemAttr, bits [3:0]

When the component supports configuring the memory type for message signaled interrupts:

Memory type. Defines the memory type and attributes for message signaled interrupts.

MemAttr	Meaning
0b0000	Device-nGnRnE memory.
0b0001	Device-nGnRE memory.
0b0010	Device-nGRE memory.
0b0011	Device-GRE memory.
0b0101	Normal memory, Inner Non-cacheable, Outer Non-cacheable.
0b0110	Normal memory, Inner Write-Through, Outer Non-cacheable.
0b0111	Normal memory, Inner Write-Back, Outer Non-cacheable.
0b1001	Normal memory, Inner Non-cacheable, Outer Write-Through.
0b1010	Normal memory, Inner Write-Through, Outer Write-Through.
0b1011	Normal memory, Inner Write-Back, Outer Write-Through.
0b1101	Normal memory, Inner Non-cacheable, Outer Write-Back.
0b1110	Normal memory, Inner Write-Through, Outer Write-Back.
0b1111	Normal memory, Inner Write-Back, Outer Write-Back.

All other values are reserved.

Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Memory type.

The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

When the implementation does not use the recommended layout for the ERRIRQCR registers:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMPLEMENTATION DEFINED																															

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRFHICR2

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRFHICR2 are IMPLEMENTATION DEFINED .

This section shows the offset of ERRFHICR2 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRFHICR2.

ERRFHICR2 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xE8C	ERRFHICR2

Accessible as follows:

- When the implementation uses message-signaled interrupts, (an access is Non-secure or an access is Realm), the implementation uses the recommended layout for the ERRIRQCR registers, and ERRFHICR2.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRGSR<m>, Error Group <n> Status Register, n = 0 - 13

The ERRGSR<m> characteristics are:

Purpose

Shows the status for the records in the group.

Configuration

ERRGSR is implemented only as part of a memory-mapped group of error records.

If FEAT_RASSA_4KB_GRP is implemented, then a single ERRGSR register is implemented.

Attributes

ERRGSR<m> is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
S63	S62	S61	S60	S59	S58	S57	S56	S55	S54	S53	S52	S51	S50	S49	S48	S47	S46	S45	S44	S43	S42	S41	S40	S39	S38	S37	S36	S35	S34	S33
S31	S30	S29	S28	S27	S26	S25	S24	S23	S22	S21	S20	S19	S18	S17	S16	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

S<n>, bit [n], for n = 63 to 0

If FEAT_RASSA_4KB_GRP is implemented, the status for error record <n>. A read-only copy of ERR<n>STATUS.V.

If FEAT_RASSA_16KB_GRP is implemented or FEAT_RASSA_64KB_GRP is implemented, the status for error record <m×64+n>. A read-only copy of ERR<m×64+n>STATUS.V.

S<n>	Meaning
0b0	No error.
0b1	One or more errors.

Accessing this field has the following behavior:

- Access to this field is **RES0** if all the following are true:
  - FEAT_RASSA_4KB_GRP is implemented.
  - n >= 56.
- Access to this field is **RES0** if all the following are true:
  - FEAT_RASSA_4KB_GRP is implemented.
  - the Common Fault Injection Model is implemented by any error record in the group.
  - n >= 24.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_RASSA_4KB_GRP is implemented.
  - error record n is not implemented.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_RASSA_16KB_GRP is implemented or FEAT_RASSA_64KB_GRP is implemented.
  - error record (m * 64) + n is not implemented.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_RASSAv2 is not implemented.
  - error record n does not support this type of reporting.
- Otherwise, access to this field is **RO**.

Accessing ERRGSR<m>

This section shows the offset of ERRGSR when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRGSR<n>.

ERRGSR<m> can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0xE00` + (64 * m)	ERRGSR<m>

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRIIDR, Implementation Identification Register

The ERRIIDR characteristics are:

Purpose

Defines the implementer of the component.

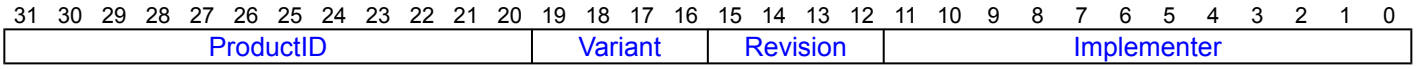
Configuration

This register is present only when RAS System Architecture v1p1 is implemented. Otherwise, direct accesses to ERRIIDR are RES0.

Attributes

ERRIIDR is a 32-bit register.

Field descriptions



ProductID, bits [31:20]

Part number, bits [11:0]. The part number is selected by the designer of the component.

If ERRPIDR0 and ERRPIDR1 are implemented, ERRPIDR0.PART_0 matches bits [7:0] of ERRIIDR.ProductID and ERRPIDR1.PART_1 matches bits [11:8] of ERRIIDR.ProductID.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [19:16]

Component major revision.

This field distinguishes product variants or major revisions of the product.

If ERRPIDR2 is implemented, ERRPIDR2.REVISION matches ERRIIDR.Variant.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Component minor revision.

This field distinguishes minor revisions of the product.

If ERRPIDR3 is implemented, ERRPIDR3.REVAND matches ERRIIDR.Revision.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the RAS component.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for ERRIIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

If ERRPIDR4 is implemented, ERRPIDR4.DES_2 matches bits [11:8] of this field.

If ERRPIDR2 is implemented, ERRPIDR2.DES_1 matches bits [6:4] of this field.

If ERRPIDR1 is implemented, ERRPIDR1.DES_0 matches bits [3:0] of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRIIDR

This section shows the offset of ERRIIDR when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRIIDR.

ERRIIDR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0xE10`	ERRIIDR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRIMPDEF<n>, IMPLEMENTATION DEFINED Register <n>, n = 0 - 191

The ERRIMPDEF<n> characteristics are:

Purpose

IMPLEMENTATION DEFINED RAS extensions.

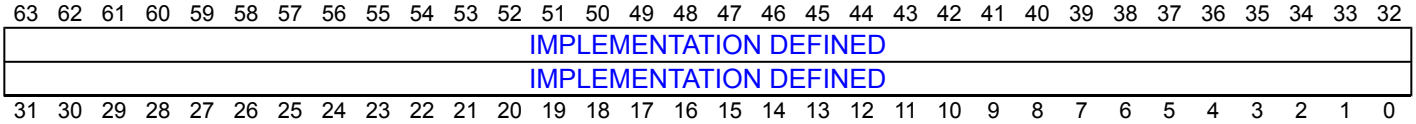
Configuration

This register is present only when the Common Fault Injection Model Extension is not implemented, UInt(ERRDEVID.NUM) <= 32, and an implementation implements ERRIMPDEF<n>. Otherwise, direct accesses to ERRIMPDEF<n> are RES0.

Attributes

ERRIMPDEF<n> is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRIMPDEF<n>

This section shows the offset of ERRIMPDEF<n> when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRIMPDEF<n>.

ERRIMPDEF<n> can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0x800` + (8 * n)	ERRIMPDEF<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRIRQCR<n>, Generic Error Interrupt Configuration Register <n>, n = 0 - 15

The ERRIRQCR<n> characteristics are:

Purpose

The ERRIRQCR<n> registers are reserved for IMPLEMENTATION DEFINED interrupt configuration registers.

The architecture provides a recommended layout for the ERRIRQCR<n> registers. These registers are named:

- ERRFHICR0, ERRFHICR1, and ERRFHICR2 for the fault handling interrupt controls.
- ERRERICR0, ERRERICR1, and ERRERICR2 for the error recovery interrupt controls.
- ERRCRICR0, ERRCRICR1, and ERRCRICR2 for the critical error interrupt controls.
- ERRIRQSR for the status register.

This section describes the generic, IMPLEMENTATION DEFINED, format.

Configuration

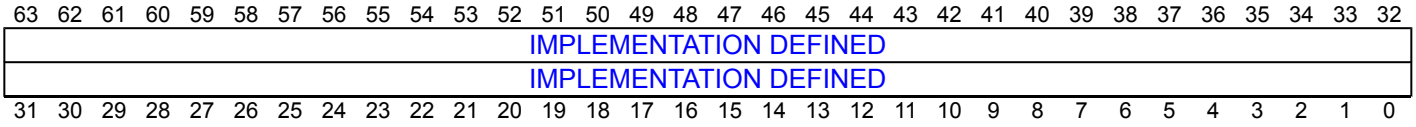
This register is present only when the interrupt configuration registers are implemented. Otherwise, direct accesses to ERRIRQCR<n> are RES0.

ERRIRQCR<n> is implemented only as part of a memory-mapped group of error records.

Attributes

ERRIRQCR<n> is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED controls. The content of these registers is IMPLEMENTATION DEFINED.

Accessing ERRIRQCR<n>

This section shows the offset of ERRIRQCR<n> when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRIRQCR<n>.

ERRIRQCR<n> can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0xE80` + (8 * n)	ERRIRQCR<n>

Accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRIRQSR, Error Interrupt Status Register

The ERRIRQSR characteristics are:

Purpose

Interrupt status register.

Configuration

This register is present only when interrupt configuration registers are implemented. Otherwise, direct accesses to ERRIRQSR are RES0.

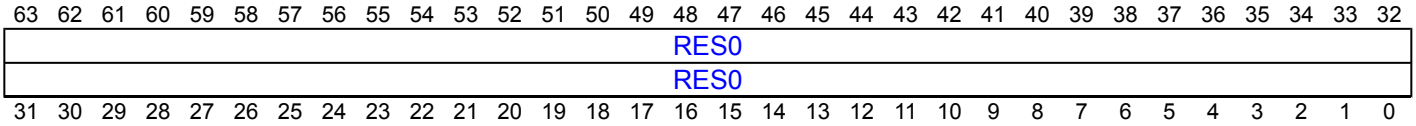
ERRIRQSR is implemented only as part of a memory-mapped group of error records.

Attributes

ERRIRQSR is a 64-bit register.

Field descriptions

When the implementation uses the recommended layout for the ERRIRQCR registers and the implementation uses simple interrupts:

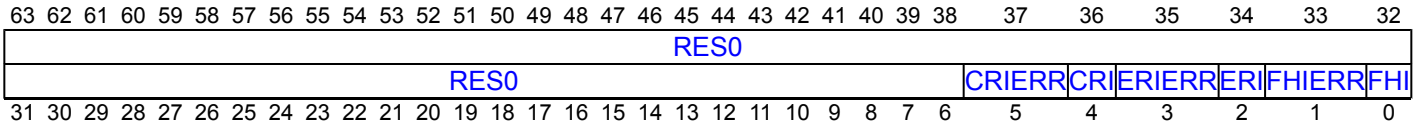


Bits [63:0]

Reserved, RES0.

To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

When the implementation uses message-signaled interrupts and the implementation uses the recommended layout for the ERRIRQCR registers:



Bits [63:6]

Reserved, RES0.

CRIERR, bit [5]

When the Critical Error Interrupt is implemented:

Critical Error Interrupt Error.

CRIERR	Meaning
0b0	Critical Error Interrupt write has not returned an error since this field was last cleared to zero.
0b1	Critical Error Interrupt write has returned an error since this field was last cleared to zero.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

CRI, bit [4]
When the Critical Error Interrupt is implemented:

Critical Error Interrupt write in progress.

CRI	Meaning
0b0	Critical Error Interrupt write not in progress.
0b1	Critical Error Interrupt write in progress.

Software must not disable an interrupt whilst the write is in progress.

Note

This field does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

ERIERR, bit [3]
When the Error Recovery Interrupt is implemented:

Error Recovery Interrupt Error.

ERIERR	Meaning
0b0	Error Recovery Interrupt write has not returned an error since this field was last cleared to zero.
0b1	Error Recovery Interrupt write has returned an error since this field was last cleared to zero.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

ERI, bit [2]

When the Error Recovery Interrupt is implemented:

Error Recovery Interrupt write in progress.

ERI	Meaning
0b0	Error Recovery Interrupt write not in progress.
0b1	Error Recovery Interrupt write in progress.

Software must not disable an interrupt whilst the write is in progress.

Note

This field does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

FHIERR, bit [1]

When the Fault Handling Interrupt is implemented:

Fault Handling Interrupt Error.

FHIERR	Meaning
0b0	Fault Handling Interrupt write has not returned an error since this field was last cleared to zero.
0b1	Fault Handling Interrupt write has returned an error since this field was last cleared to zero.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

FHI, bit [0]

When the Fault Handling Interrupt is implemented:

Fault Handling Interrupt write in progress.

FHI	Meaning
0b0	Fault Handling Interrupt write not in progress.
0b1	Fault Handling Interrupt write in progress.

Software must not disable an interrupt whilst the write is in progress.

Note

This field does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

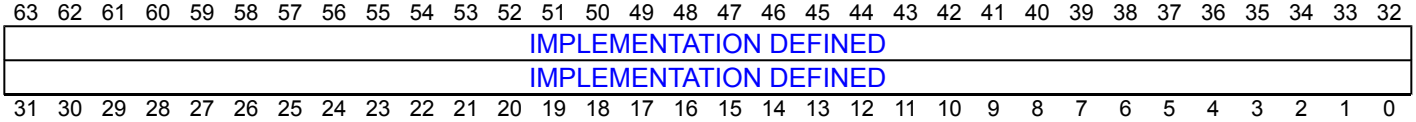
To determine whether an interrupt is active, software must examine the individual ERR<n>STATUS registers.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR registers:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRIRQSR

If the implementation does not use the recommended layout for the ERRIRQCR registers then accesses to ERRIRQSR are IMPLEMENTATION DEFINED.

This section shows the offset of ERRIRQSR when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRIRQSR.

ERRIRQSR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xEF8	ERRIRQSR

Accessible as follows:

- When the implementation uses message-signaled interrupts, (an access is Non-secure or an access is Realm), the implementation uses the recommended layout for the ERRIRQCR registers, and ERRIRQSR.NSMSI configures the physical address space for message-signaled interrupts as Secure, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>ADDR, Error Record <n> Address Register, n = 0 - 65534

The ERR<n>ADDR characteristics are:

Purpose

If an address is associated with a detected error, then it is written to ERR<n>ADDR when the error is recorded. It is IMPLEMENTATION DEFINED how the recorded address maps to the software-visible physical address. Software might have to reconstruct the actual physical addresses using the identity of the node and knowledge of the system.

Configuration

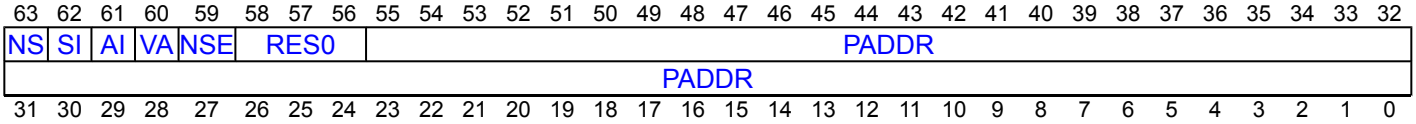
This register is present only when FEAT_RAS is implemented, error record n is implemented, and error record n includes an address associated with an error. Otherwise, direct accesses to ERR<n>ADDR are RES0.

ERRFR[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record then FirstRecordOfNode(n) = n.

Attributes

ERR<n>ADDR is a 64-bit register.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

Non-secure attribute. With ERR<n>ADDR.NSE, indicates the physical address space of the recorded location.

NS	Meaning
0b0	When ERR<n>ADDR.NSE == 0: ERR<n>ADDR.PADDR is a Secure address. When ERR<n>ADDR.NSE == 1: ERR<n>ADDR.PADDR is a Root address.
0b1	When ERR<n>ADDR.NSE == 0: ERR<n>ADDR.PADDR is a Non-secure address. When ERR<n>ADDR.NSE == 1: ERR<n>ADDR.PADDR is a Realm address.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure attribute.

NS	Meaning
0b0	ERR<n>ADDR.PADDR is a Secure address.
0b1	ERR<n>ADDR.PADDR is a Non-secure address.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SI, bit [62]

When FEAT_RME is implemented:

Secure Incorrect. Indicates whether ERR<n>ADDR.{NS, NSE} are valid.

SI	Meaning
0b0	ERR<n>ADDR.{NS, NSE} are correct. That is, they match the software's view of the physical address space for the recorded location.
0b1	ERR<n>ADDR.{NS, NSE} might not be correct, and might not match the software's view of the physical address space for the recorded location.

It is IMPLEMENTATION DEFINED whether this field is read-only or read/write.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Secure Incorrect. Indicates whether ERR<n>ADDR.NS is valid.

SI	Meaning
0b0	ERR<n>ADDR.NS is correct. That is, it matches the software's view of the Non-secure attribute for the recorded location.
0b1	ERR<n>ADDR.NS might not be correct, and might not match the software's view of the Non-secure attribute for the recorded location.

It is IMPLEMENTATION DEFINED whether this field is read-only or read/write.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

AI, bit [61]

Address Incorrect. Indicates whether ERR<n>ADDR.PADDR is a valid physical address that is known to match the software's view of the physical address for the recorded location.

AI	Meaning
0b0	ERR<n>ADDR.PADDR is a valid physical address. That is, it matches the software's view of the physical address for the recorded location.
0b1	ERR<n>ADDR.PADDR might not be a valid physical address, and might not match the software's view of the physical address for the recorded location.

It is IMPLEMENTATION DEFINED whether this field is read-only or read/write.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

VA, bit [60]

Virtual Address. Indicates whether ERR<n>ADDR.PADDR field is a virtual address.

VA	Meaning
0b0	ERR<n>ADDR.PADDR is not a virtual address.
0b1	ERR<n>ADDR.PADDR is a virtual address.

No context information is provided for the virtual address. When ERR<n>ADDR.VA is recorded as 1, ERR<n>ADDR.{NS, SI, AI} are recorded as {0, 1, 1} and, if FEAT_RME is implemented, ERR<n>ADDR.NSE is recorded as 0.

Support for this field is optional. If this field is not implemented and ERR<n>ADDR.PADDR field is a virtual address, then ERR<n>ADDR.{NS, SI, AI} read as {0, 1, 1} and, if FEAT_RME is implemented, ERR<n>ADDR.NSE reads as 0.

It is IMPLEMENTATION DEFINED whether this field is read-only or read/write.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [59]
When FEAT_RME is implemented:

Physical Address Space. Together with ERR<n>ADDR.NS, indicates the address space for ERR<n>ADDR.PADDR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [58:56]

Reserved, RES0.

PADDR, bits [55:0]

Physical Address. Address of the recorded location. If the physical address size implemented by this component is smaller than the size of this field, then high-order bits are unimplemented and either RES0 or have a fixed read-only IMPLEMENTATION DEFINED value. Low-order address bits might also be unimplemented and RES0, for example, if the physical address is always aligned to the size of a protection granule.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing ERR<n>ADDR

ERR<n>ADDR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0x018` + (64 * n)	ERR<n>ADDR

Accessible as follows:

- When the node that owns error record n implements the Common Fault Injection Model Extension, ERRPFGF[FirstRecordOfNode(n)].AV == 0, and ERR<n>STATUS.AV == 1, accesses to this register are **RO**.
- When the node that owns error record n does not implement the Common Fault Injection Model Extension and ERR<n>STATUS.AV == 1, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>CTLR, Error Record <n> Control Register, n = 0 - 65534

The ERR<n>CTLR characteristics are:

Purpose

The error control register contains enable bits for the node that writes to this record:

- Enabling error detection and correction.
- Enabling the critical error, error recovery, and fault handling interrupts.
- Enabling in-band error response for uncorrected errors.

For each bit, if the node does not support the feature, then the bit is RES0. The definition of each record is IMPLEMENTATION DEFINED.

Configuration

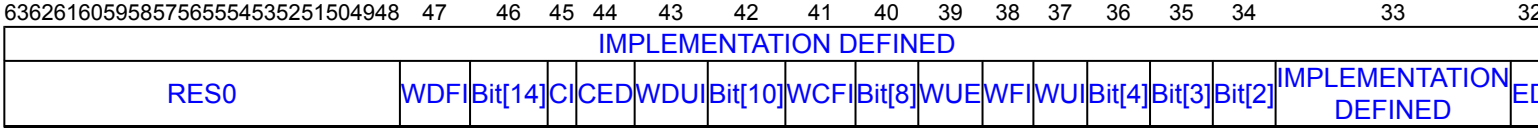
This register is present only when FEAT_RAS is implemented, error record n is implemented, and error record n is the first error record in the node. Otherwise, direct accesses to ERR<n>CTLR are RES0.

ERR<n>FR contains additional information about the node.

Attributes

ERR<n>CTLR is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:32]

Reserved for IMPLEMENTATION DEFINED controls. Must permit SBZP write policy for software.

Bits [31:16]

Reserved, RES0.

WDFI, bit [15]

When RAS System Architecture v2 is implemented and ERR<n>FR.DFI == 0b11:

Fault handling interrupt for Deferred errors on writes enable, with ERR<n>CTLR.WFI.

When enabled by ERR<n>CTLR.{WDFI, `WFI`}:

- The fault handling interrupt is generated for errors recorded as Deferred error on writes.
- If the corresponding fault handling interrupt control for corrected error events, ERR<n>CTLR.WCFI, is not implemented, then the fault handling interrupt is generated for corrected error events on writes.

WDFI	Meaning
0b0	When ERR<n>CTLR.WFI == 0, Fault handling interrupt not generated for Deferred errors on writes. When ERR<n>CTLR.WFI == 1, Fault handling interrupt generated for Deferred errors on writes.
0b1	When ERR<n>CTLR.WFI == 0, Fault handling interrupt generated for Deferred errors on writes. When ERR<n>CTLR.WFI == 1, Fault handling interrupt not generated for Deferred errors on writes.

See ERR<n>CTLR.CFI for more information on corrected error events.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[14]
When RAS System Architecture v2 is implemented and ERR<n>FR.DFI == 0b10:

DFI, bit [14]

Fault handling interrupt for Deferred errors enable, with ERR<n>CTLR.FI.

When ERR<n>FR.DFI == 0b10, this control applies to errors on both reads and writes.

When enabled by ERR<n>CTLR.{DFI, FI}:

- The fault handling interrupt is generated for all errors recorded as Deferred error.
- If the fault handling interrupt control for corrected error events, ERR<n>CTLR.CFI, is not implemented, then the fault handling interrupt is generated for all corrected error events.

DFI	Meaning
0b0	When ERR<n>CTLR.FI == 0, Fault handling interrupt not generated for Deferred errors. When ERR<n>CTLR.FI == 1, Fault handling interrupt generated for Deferred errors.
0b1	When ERR<n>CTLR.FI == 0, Fault handling interrupt generated for Deferred errors. When ERR<n>CTLR.FI == 1, Fault handling interrupt not generated for Deferred errors.

See ERR<n>CTLR.CFI for more information on corrected error events.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When RAS System Architecture v2 is implemented and ERR<n>FR.DFI == 0b11:

RDFI, bit [14]

Fault handling interrupt for Deferred errors on reads enable, with ERR<n>CTLR.RFI.

When enabled by ERR<n>CTLR.{RDFI, RFI}:

- The fault handling interrupt is generated for errors recorded as Deferred error on reads.

- If the corresponding fault handling interrupt control for corrected error events, ERR<n>CTLR.RCFI, is not implemented, then the fault handling interrupt is generated for corrected error events on reads.

RDFI	Meaning
0b0	When ERR<n>CTLR.RFI == 0, Fault handling interrupt not generated for Deferred errors on reads. When ERR<n>CTLR.RFI == 1, Fault handling interrupt generated for Deferred errors on reads.
0b1	When ERR<n>CTLR.RFI == 0, Fault handling interrupt generated for Deferred errors on reads. When ERR<n>CTLR.RFI == 1, Fault handling interrupt not generated for Deferred errors on reads.

See ERR<n>CTLR.CFI for more information on corrected error events.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CI, bit [13]
When ERR<n>FR.CI == 0b10:

Critical error interrupt enable. When enabled, the critical error interrupt is generated for a critical error condition.

CI	Meaning
0b0	Critical error interrupt not generated for critical errors. Critical errors are treated as Uncontained errors.
0b1	Critical error interrupt generated for critical errors.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CED, bit [12]
When RAS System Architecture v2 is implemented, ERR<n>FR.CEC != 0b000, and ERR<n>FR.CED == 1:

Disable generation of corrected error events from error counters.

CED	Meaning
0b0	Corrected error events are generated by the error counter or counters.
0b1	Corrected error events are generated when a Corrected error is recorded.

See ERR<n>CTLR.CFI for more information on corrected error events.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WDUI, bit [11]

When ERR<n>FR.DUI == 0b11:

Error recovery interrupt for Deferred errors on writes enable.

When enabled, the error recovery interrupt is generated for errors recorded as Deferred error on writes.

WDUI	Meaning
0b0	Error recovery interrupt not generated for Deferred errors on writes.
0b1	Error recovery interrupt generated for Deferred errors on writes.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[10]

When ERR<n>FR.DUI == 0b10:

DUI, bit [10]

Error recovery interrupt for Deferred errors enable.

When ERR<n>FR.DUI == 0b10, this control applies to errors arising from both reads and writes.

When enabled, the error recovery interrupt is generated for all errors recorded as Deferred error.

DUI	Meaning
0b0	Error recovery interrupt not generated for Deferred errors.
0b1	Error recovery interrupt generated for Deferred errors.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.DUI == 0b11:

RDUI, bit [10]

Error recovery interrupt for Deferred errors on reads enable.

When enabled, the error recovery interrupt is generated for errors recorded as Deferred error on reads.

RDUI	Meaning
0b0	Error recovery interrupt not generated for Deferred errors on reads.
0b1	Error recovery interrupt generated for Deferred errors on reads.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WCFI, bit [9]
When ERR<n>FR.CFI == 0b11:

Fault handling interrupt for corrected error events on writes enable.

When enabled, the fault handling interrupt is generated for corrected error events on writes.

WCFI	Meaning
0b0	Fault handling interrupt not generated for corrected error events on writes.
0b1	Fault handling interrupt generated for corrected error events on writes.

See ERR<n>CTLR.CFI for more information on corrected error events.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[8]
When ERR<n>FR.CFI == 0b10:

CFI, bit [8]

Fault handling interrupt for corrected error events enable.

When ERR<n>FR.CFI == 0b10, this control applies to errors on both reads and writes.

When enabled, the fault handling interrupt is generated for all corrected error events.

CFI	Meaning
0b0	Fault handling interrupt not generated for corrected error events.
0b1	Fault handling interrupt generated for corrected error events.

If the node implements a corrected error counter or counters, and either ERR<n>CTLR.CED is not implemented or ERR<n>CTLR.CED is 0, then a corrected error event is defined as follows:

- A corrected error event occurs when a counter overflows and sets a counter overflow flag to 1.
- It is UNPREDICTABLE whether a corrected error event occurs when a software write sets a counter overflow flag to 1.
- It is UNPREDICTABLE whether a corrected error event occurs when a counter overflows and the overflow flag was previously set to 1.

Otherwise, a corrected error event occurs when the error record records an error as a Corrected error.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.CFI == 0b11:

RCFI, bit [8]

Fault handling interrupt for corrected error events on reads enable.

When enabled, the fault handling interrupt is generated for corrected error events on reads.

RCFI	Meaning
0b0	Fault handling interrupt not generated for corrected error events on reads.
0b1	Fault handling interrupt generated for corrected error events on reads.

If the node implements a corrected error counter or counters, and either ERR<n>CTLR.CED is not implemented or ERR<n>CTLR.CED is 0, then a corrected error event is defined as follows:

- A corrected error event occurs when a counter overflows and sets a counter overflow flag to 1.
- It is UNPREDICTABLE whether a corrected error event occurs when a software write sets a counter overflow flag to 1.
- It is UNPREDICTABLE whether a corrected error event occurs when a counter overflows and the overflow flag was previously set to 1.

Otherwise, a corrected error event occurs when the error record records an error as a Corrected error.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WUE, bit [7]
When ERR<n>FR.UE == 0b11:

In-band error response on writes enable.

When enabled, responses to writes that detect an error that is not corrected and is not deferred are signaled with an in-band error response (External abort).

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

WUE	Meaning
0b0	In-band error response for uncorrected errors on writes disabled.
0b1	In-band error response for uncorrected errors on writes enabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WFI, bit [6]
When ERR<n>FR.FI == 0b11:

Fault handling interrupt on writes enable.

When enabled:

- The fault handling interrupt is generated for errors recorded as Uncorrected error on writes.
- If the corresponding fault handling interrupt control for Deferred errors, ERR<n>CTLR.WDFI, is not implemented, then the fault handling interrupt is generated for errors recorded as Deferred error on writes.
- If the corresponding fault handling interrupt controls for Deferred errors and corrected error events, ERR<n>CTLR.{WDFI, WCFI}, are not implemented, then the fault handling interrupt is generated for corrected error events on writes.

WFI	Meaning
0b0	Fault handling interrupt on writes disabled.
0b1	Fault handling interrupt on writes enabled.

See ERR<n>CTLR.CFI for more information on corrected error events.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WUI, bit [5]

When ERR<n>FR.UI == 0b11:

Uncorrected error recovery interrupt on writes enable.

When enabled, the error recovery interrupt is generated for errors recorded as Uncorrected error on writes.

WUI	Meaning
0b0	Error recovery interrupt on writes disabled.
0b1	Error recovery interrupt on writes enabled.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[4]

When ERR<n>FR.UE == 0b10:

UE, bit [4]

In-band error response enable.

When ERR<n>FR.UE == 0b10, this control applies to errors arising from both reads and writes.

When enabled, responses to transactions that detect an error that is not corrected and is not deferred are signaled with an in-band error response (External abort).

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

UE	Meaning
0b0	In-band error response for uncorrected errors disabled.
0b1	In-band error response for uncorrected errors enabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.UE == 0b11:

RUE, bit [4]

In-band error response on reads enable.

When enabled, responses to reads that detect an error that is not corrected and is not deferred are signaled with an in-band error response (External abort).

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

RUE	Meaning
0b0	In-band error response for uncorrected errors on reads disabled.
0b1	In-band error response for uncorrected errors on reads enabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[3]

When ERR<n>FR.FI == 0b10:

FI, bit [3]

Fault handling interrupt enable.

When ERR<n>FR.FI == 0b10, this control applies to errors on both reads and writes.

When enabled:

- The fault handling interrupt is generated for all errors recorded as Uncorrected error.
- If the fault handling interrupt control for Deferred errors, ERR<n>CTLR.DFI, is not implemented, then the fault handling interrupt is generated for all errors recorded as Deferred error.
- If the fault handling interrupt controls for Deferred errors and corrected error events, ERR<n>CTLR.{DFI, CFI}, are not implemented, then the fault handling interrupt is generated for all corrected error events.

FI	Meaning
0b0	Fault handling interrupt disabled.
0b1	Fault handling interrupt enabled.

See ERR<n>CTLR.CFI for more information on corrected error events.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.FI == 0b11:

RFI, bit [3]

Fault handling interrupt on reads enable.

When enabled:

- The fault handling interrupt is generated for errors recorded as Uncorrected error on reads.
- If the corresponding fault handling interrupt control for Deferred errors, ERR<n>CTLR.RDFI, is not implemented, then the fault handling interrupt is generated for errors recorded as Deferred error on reads.
- If the corresponding fault handling interrupt controls for Deferred errors and corrected error events, ERR<n>CTLR.{RDFI, RCFI}, are not implemented, then the fault handling interrupt is generated for corrected error events on reads.

RFI	Meaning
0b0	Fault handling interrupt on reads disabled.
0b1	Fault handling interrupt on reads enabled.

See ERR<n>CTLR.CFI for more information on corrected error events.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit[2]

When ERR<n>FR.UI == 0b10:

UI, bit [2]

Uncorrected error recovery interrupt enable.

When ERR<n>FR.UI == 0b10, this control applies to errors arising from both reads and writes.

When enabled, the error recovery interrupt is generated for all errors recorded as Uncorrected error.

UI	Meaning
0b0	Error recovery interrupt disabled.
0b1	Error recovery interrupt enabled.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.UI == 0b11:

RUI, bit [2]

Uncorrected error recovery interrupt on reads enable.

When enabled, the error recovery interrupt is generated for errors recorded as Uncorrected error on reads.

RUI	Meaning
0b0	Error recovery interrupt on reads disabled.
0b1	Error recovery interrupt on reads enabled.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IMPLEMENTATION DEFINED, bit [1]

Reserved for IMPLEMENTATION DEFINED controls. Must permit SBZP write policy for software.

ED, bit [0]

When ERR<n>FR.ED == 0b10:

Error reporting and logging enable. When disabled, the node behaves as if error detection and correction are disabled, and no errors are recorded or signaled by the node. Arm recommends that, when disabled, correct error detection and correction codes are written for writes, unless disabled by an IMPLEMENTATION DEFINED control for error injection.

ED	Meaning
0b0	Error reporting disabled.
0b1	Error reporting enabled.

It is IMPLEMENTATION DEFINED whether the node fully disables error detection and correction when reporting is disabled. That is, even with error reporting disabled, the node might continue to silently correct errors. Uncorrected errors might result in corrupt data being silently propagated by the node.

Note

If this node requires initialization after Cold reset to prevent signaling false errors, then Arm recommends this field is set to 0 on Cold reset, meaning errors are not reported from Cold reset. This allows boot software to initialize a node without signaling errors. Software can enable error reporting after the node is initialized. Otherwise, the Cold reset value is IMPLEMENTATION DEFINED. If the Cold reset value is 1, the reset values of other controls in this register are also IMPLEMENTATION DEFINED and should not be UNKNOWN.

The reset behavior of this field is:

- On an Error recovery reset, when RAS System Architecture v2 is implemented and ERR<n>FR.SRV == 1, this field resets to '0'.
- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERR<n>FR.SRV == 1, this field resets to '0'.
  - Otherwise, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

Accessing ERR<n>CTLR

ERR<n>CTLR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0x008 + (64 * n)	ERR<n>CTLR

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>FR, Error Record <n> Feature Register, n = 0 - 65534

The ERR<n>FR characteristics are:

Purpose

Defines whether error record <n> is the first record owned by a node:

- If error record <n> is the first error record owned by a node, then ERR<n>FR.ED is not `0b00`.
- If error record <n> is not the first error record owned by a node, then ERR<n>FR.ED is `0b00`.

If error record <n> is the first record owned by the node, defines which of the common architecturally-defined features are implemented by the node and, of the implemented features, which are software programmable.

Configuration

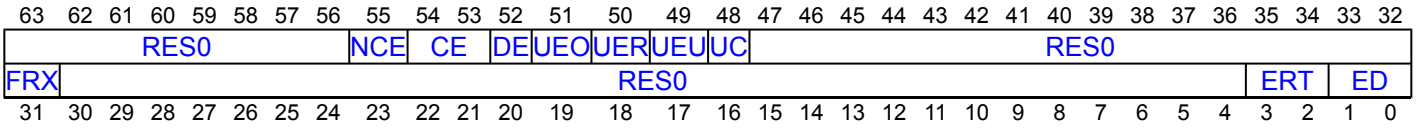
This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERR<n>FR are RES0.

Attributes

ERR<n>FR is a 64-bit register.

Field descriptions

When error record n is not implemented or error record n is not the first error record in the node:



Bits [63:56]

Reserved, RES0.

NCE, bit [55]

When ERR<n>FR.FRX == 1 and ERRFR[FirstRecordOfNode(n)].CEC != 0b000:

No countable errors. Describes whether this error record supports recording countable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCE	Meaning
`0b0`	Records countable errors.
`0b1`	Does not record countable errors.

When ERRFR[FirstRecordOfNode(n)].CEC != `0b000`, at least one error record owned by the node records countable errors.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

CE, bits [54:53]

When ERR<n>FR.FRX == 1:

Corrected Error recording. Describes the types of Corrected errors the error record can record, if any.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CE	Meaning
0b00	Does not record Corrected errors.
0b01	Records only transient or persistent Corrected errors. That is, Corrected errors recorded by setting ERR<n>STATUS.CE to either 0b01 or 0b11.
0b10	Records only non-specific Corrected errors. That is, Corrected errors recorded by setting ERR<n>STATUS.CE to 0b10.
0b11	Records all types of Corrected error.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

DE, bit [52]

When ERR<n>FR.FRX == 1:

Deferred Error recording. Describes whether the error record supports recording Deferred errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DE	Meaning
0b0	Does not record Deferred errors.
0b1	Records Deferred errors.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

UEO, bit [51]

When ERR<n>FR.FRX == 1:

Latent or Restartable Error recording. Describes whether the error record supports recording Latent or Restartable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEO	Meaning
0b0	Does not record Latent or Restartable errors.
0b1	Records Latent or Restartable errors.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

UER, bit [50]

When ERR<n>FR.FRX == 1:

Signaled or Recoverable Error recording. Describes whether the error record supports recording Signaled or Recoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UER	Meaning
0b0	Does not record Signaled or Recoverable errors.
0b1	Records Signaled or Recoverable errors.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

UEU, bit [49]

When ERR<n>FR.FRX == 1:

Unrecoverable Error recording. Describes whether the error record supports recording Unrecoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEU	Meaning
0b0	Does not record Unrecoverable errors.
0b1	Records Unrecoverable errors.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

UC, bit [48]

When ERR<n>FR.FRX == 1:

Uncontainable Error recording. Describes whether the error record supports recording Uncontainable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UC	Meaning
0b0	Does not record Uncontainable errors.
0b1	Records Uncontainable errors.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [47:32]

Reserved, RES0.

FRX, bit [31]

When error record n is implemented, RAS System Architecture v2 is implemented, and ERR<n>FR.ERT == 0b00:

Feature Register extension. Defines whether ERR<n>FR[63:48] describe architecturally-defined properties of this error record, including the supported error types.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FRX	Meaning
0b0	ERR<n>FR[63:48] are RES0.
0b1	ERR<n>FR[63:48] are defined by the architecture.

If ERR<n>FR.FRX is 0, error record <n> is implemented, and ERRFR[FirstRecordOfNode(n)].FRX is 1, then ERRFR[FirstRecordOfNode(n)][63:48] describe the architecturally-defined properties of all error records owned by the node.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [30:4]

Reserved, RES0.

ERT, bits [3:2]

When RAS System Architecture v2 is implemented:

Error Record Type. Defines the type of error record.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ERT	Meaning
0b00	Error record <n> not implemented or is a normal record that is not the first error record of the node.
0b01	Error record <n> is a continuation record of the previous error record, <n-1>.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

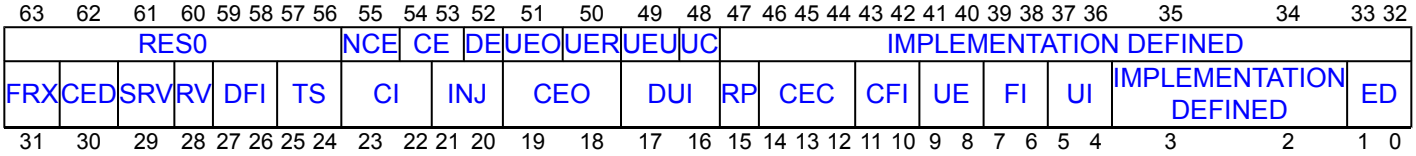
ED, bits [1:0]

Error reporting and logging. Indicates error record <n> is not the first error record owned the node.

ED	Meaning
0b00	Error record <n> is not implemented or is not the first error record owned by the node.

Access to this field is **RO**.

When error record n is the first error record in the node:



Bits [63:56]

When ERR<n>FR.FRX == 1:

Reserved, RES0.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

NCE, bit [55]

When RAS System Architecture v2 is implemented, ERR<n>FR.FRX == 1, and ERR<n>FR.CEC != 0b000:

No countable errors. Describes whether this error record supports recording countable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCE	Meaning
0b0	Records countable errors.
0b1	Does not record countable errors.

When ERR<n>FR.CEC != 0b000, at least one error record owned by the node records countable errors.

Access to this field is **RO**.

When ERR<n>FR.FRX == 1:

Reserved, RES0.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

CE, bits [54:53]

When ERR<n>FR.FRX == 1:

Corrected Error recording. Describes the types of Corrected errors the node can record, if any.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CE	Meaning
0b00	Does not record Corrected errors.
0b01	Records only transient or persistent Corrected errors. That is, Corrected errors recorded by setting ERR<n>STATUS.CE to either 0b01 or 0b11.
0b10	Records only non-specific Corrected errors. That is, Corrected errors recorded by setting ERR<n>STATUS.CE to 0b10.
0b11	Records all types of Corrected error.

Access to this field is **RO**.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

DE, bit [52]

When ERR<n>FR.FRX == 1:

Deferred Error recording. Describes whether the node supports recording Deferred errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DE	Meaning
0b0	Does not record Deferred errors.
0b1	Records Deferred errors.

Access to this field is **RO**.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UEO, bit [51]

When ERR<n>FR.FRX == 1:

Latent or Restartable Error recording. Describes whether the node supports recording Latent or Restartable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEO	Meaning
0b0	Does not record Latent or Restartable errors.
0b1	Records Latent or Restartable errors.

Access to this field is **RO**.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UER, bit [50]

When ERR<n>FR.FRX == 1:

Signaled or Recoverable Error recording. Describes whether the node supports recording Signaled or Recoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UER	Meaning
0b0	Does not record Signaled or Recoverable errors.
0b1	Records Signaled or Recoverable errors.

Access to this field is **RO**.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UEU, bit [49]

When ERR<n>FR.FRX == 1:

Unrecoverable Error recording. Describes whether the node supports recording Unrecoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEU	Meaning
0b0	Does not record Unrecoverable errors.
0b1	Records Unrecoverable errors.

Access to this field is **RO**.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UC, bit [48]

When ERR<n>FR.FRX == 1:

Uncontainable Error recording. Describes whether the node supports recording Uncontainable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UC	Meaning
0b0	Does not record Uncontainable errors.
0b1	Records Uncontainable errors.

Access to this field is **RO**.

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

IMPLEMENTATION DEFINED, bits [47:32]

Reserved for identifying IMPLEMENTATION DEFINED controls.

FRX, bit [31]

When RAS System Architecture v1p1 is implemented:

Feature Register extension.

Defines whether ERR<n>FR[63:48] describe architecturally-defined properties of this error record or node, including the supported error types, or describe IMPLEMENTATION DEFINED properties.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FRX	Meaning
0b0	ERR<n>FR[63:48] are IMPLEMENTATION DEFINED.
0b1	ERR<n>FR[63:48] are defined by the architecture.

When ERR<n>FR.FRX is 1:

- If RAS System Architecture v2 is implemented and ERR<m>FR.FRX is 1 for other error records <m> owned by the same node, then ERR<n>FR[63:48] describe the architecturally-defined properties of error record <n> only, and ERR<m>FR[63:48] describe the properties for error record <m>.
- Otherwise, ERR<n>FR[63:48] describe the architecturally-defined properties of all error records owned by the node.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

CED, bit [30]

When RAS System Architecture v2 is implemented and ERR<n>FR.CEC != 0b000:

Error counter disable. Indicates whether the node implements a control to disable any implemented Corrected error counters.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CED	Meaning
0b0	Error counter disable control is not implemented and the error counter(s) are always enabled. ERR<n>CTLR.CED is RES0.
0b1	Enabling and disabling of error counter(s) is supported and controlled by ERR<n>CTLR.CED.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

SRV, bit [29]

When RAS System Architecture v2 is implemented:

Status Reset Value. Indicates how node <n> and each error record <m> owned by node <n> is reset.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SRV	Meaning
0b0	Node <n> and each error record <m> owned by node <n> are reset as follows: • ERR<m>STATUS.{AV, V, MV} are set to {0, 0, 0} on a Cold reset and preserved on Error Recovery reset. • ERR<n>CTLR.ED is set to an IMPLEMENTATION DEFINED value on a Cold reset and preserved on Error Recovery reset.
0b1	Node <n> and each error record <m> owned by node <n> are reset as follows: • ERR<m>STATUS.{AV, V, MV} are set to architecturally UNKNOWN values on a Cold reset and preserved on Error Recovery reset. • ERR<n>CTLR.ED is set to 0 on both Cold reset and Error Recovery reset.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

RV, bit [28]

When RAS System Architecture v2 is implemented:

Reset Valid. Indicates whether each error record <m> implemented by the node includes the Reset Valid flags, ERR<m>STATUS.{RV, RV2}.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RV	Meaning
0b0	ERR<m>STATUS.{RV, RV2} are RES0.
0b1	ERR<m>STATUS.{RV, RV2} are R/W1C bits. See ERR<m>STATUS.{RV, RV2} for more information.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

DFI, bits [27:26]

When RAS System Architecture v2 is implemented and !(ERR<n>FR.FI IN {0b0x}):

Fault handling interrupt for deferred errors control. Indicates whether the enabling and disabling of fault handling interrupts on deferred errors is supported by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DFI	Meaning
0b00	Does not support the enabling and disabling of fault handling interrupts on deferred errors. ERR<n>CTLR.DFI is RES0.
0b10	Enabling and disabling of fault handling interrupts on deferred errors is supported and controllable using ERR<n>CTLR.DFI.
0b11	Enabling and disabling of fault handling interrupts on deferred errors is supported, and controllable using ERR<n>CTLR.WDFI for writes and ERR<n>CTLR.RDFI for reads.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

TS, bits [25:24]

Timestamp Extension. Indicates whether, for each error record <m> owned by this node, ERR<m>MISC3 is used as the timestamp register, and, if it is, the timebase used by the timestamp.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TS	Meaning
0b00	Does not support a timestamp register.
0b01	Implements a timestamp register in ERR<n>MISC3 for each error record <m> owned by the node. The timestamp uses the same timebase as the system Generic Timer. **Note** For an error record that has an affinity to a PE, this is the same timer that is visible through CNTPCT_EL0 at the highest Exception level on that PE.
0b10	Implements a timestamp register in ERR<m>MISC3 for each error record <m> owned by the node. The timestamp uses an IMPLEMENTATION DEFINED timebase.

All other values are reserved.

Access to this field is **RO**.

CI, bits [23:22]

Critical error interrupt. Indicates whether the critical error interrupt and associated controls are implemented by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CI	Meaning
0b00	Does not support the critical error interrupt. ERR<n>CTLR.CI is RES0.
0b01	Critical error interrupt is supported and always enabled. ERR<n>CTLR.CI is RES0.
0b10	Critical error interrupt is supported and controllable using ERR<n>CTLR.CI.

All other values are reserved.

Access to this field is **RO**.

INJ, bits [21:20]

Fault Injection Extension. Indicates whether the Common Fault Injection Model Extension is implemented by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

INJ	Meaning
0b00	Does not support the Common Fault Injection Model Extension.
0b01	Supports the Common Fault Injection Model Extension. See ERR<n>PFGF for more information.

All other values are reserved.

Access to this field is **RO**.

CEO, bits [19:18]

When ERR<n>FR.CEC != 0b000:

Corrected Error overwrite. Indicates the behavior of the node when a second or subsequent Corrected error is recorded and a first Corrected error has previously been recorded by an error record <m> owned by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CEO	Meaning
0b00	Keeps the previous error syndrome.
0b01	If ERR<m>STATUS.OF is 1 before the Corrected error is counted, then the error record keeps the previous syndrome. Otherwise the previous syndrome is overwritten.

All other values are reserved.

The second or subsequent Corrected error is counted by the Corrected error counter, regardless of the value of this field. If counting the error causes unsigned overflow of the counter, then ERR<m>STATUS.OF is set to 1.

This means that, if no other error is subsequently recorded that overwrites the syndrome:

- If ERR<n>FR.CEO is 0b00, the error record holds the syndrome for the first recorded Corrected error.
- If ERR<n>FR.CEO is 0b01, the error record holds the syndrome for the most recently recorded Corrected error before the counter overflows.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

DUI, bits [17:16]

When ERR<n>FR.UI != 0b00:

Error recovery interrupt for deferred errors control. Indicates whether the enabling and disabling of error recovery interrupts on deferred errors is supported by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DUI	Meaning
0b00	Does not support the enabling and disabling of error recovery interrupts on deferred errors. ERR<n>CTLR.DUI is RES0.
0b10	Enabling and disabling of error recovery interrupts on deferred errors is supported and controllable using ERR<n>CTLR.DUI.
0b11	Enabling and disabling of error recovery interrupts on deferred errors is supported, and controllable using ERR<n>CTLR.WDUI for writes and ERR<n>CTLR.RDUI for reads.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

RP, bit [15]

When ERR<n>FR.CEC != 0b000:

Repeat counter. Indicates whether the node implements a second Corrected error counter in ERR<m>MISC0 for each error record <m> owned by the node that can record countable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RP	Meaning
0b0	Implements a single Corrected error counter in ERR<m>MISC0 for each error record <m> owned by the node that can record countable errors.
0b1	Implements a first (repeat) counter and a second (other) counter in ERR<m>MISC0 for each error record <m> owned by the node that can record countable errors. The repeat counter is the same size as the primary error counter.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

CEC, bits [14:12]

Corrected Error Counter. Indicates whether the node implements the standard format Corrected error counter mechanisms in ERR<m>MISC0 for each error record <m> owned by the node that can record countable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CEC	Meaning
0b000	Does not implement the standard format Corrected error counter model.
0b010	Implements an 8-bit Corrected error counter in ERR<m>MISC0[39:32] for each error record <m> owned by the node that can record countable errors.
0b100	Implements a 16-bit Corrected error counter in ERR<m>MISC0[47:32] for each error record <m> owned by the node that can record countable errors.

All other values are reserved.

Note

Implementations might include other error counter models, or might include the standard format model and not indicate this in ERR<n>FR.

Access to this field is **RO**.

CFI, bits [11:10]

When !(ERR<n>FR.FI IN {0b0x}):

Fault handling interrupt for corrected errors control. Indicates whether the enabling and disabling of fault handling interrupts on corrected errors is supported by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CFI	Meaning
0b00	Does not support the enabling and disabling of fault handling interrupts on corrected errors. ERR<n>CTLR.CFI is RES0.
0b10	Enabling and disabling of fault handling interrupts on corrected errors is supported and controllable using ERR<n>CTLR.CFI.
0b11	Enabling and disabling of fault handling interrupts on corrected errors is supported, and controllable using ERR<n>CTLR.WCFI for writes and ERR<n>CTLR.RCFI for reads.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

UE, bits [9:8]

In-band error response (External abort). Indicates whether the in-band error response and associated controls are implemented by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UE	Meaning
0b00	Does not support the in-band error response. ERR<n>CTLR.UE is RES0.
0b01	In-band error response is supported and always enabled. ERR<n>CTLR.UE is RES0.
0b10	In-band error response is supported and controllable using ERR<n>CTLR.UE.
0b11	In-band error response is supported, and controllable using ERR<n>CTLR.WUE for writes and ERR<n>CTLR.RUE for reads.

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

Access to this field is **RO**.

FI, bits [7:6]

Fault handling interrupt. Indicates whether the fault handling interrupt and associated controls are implemented by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FI	Meaning
0b00	Does not support the fault handling interrupt. ERR<n>CTLR.FI is RES0.
0b01	Fault handling interrupt is supported and always enabled. ERR<n>CTLR.FI is RES0.
0b10	Fault handling interrupt is supported and controllable using ERR<n>CTLR.FI.
0b11	Fault handling interrupt is supported, and controllable using ERR<n>CTLR.WFI for writes and ERR<n>CTLR.RFI for reads.

Access to this field is **RO**.

UI, bits [5:4]

Error recovery interrupt for uncorrected errors. Indicates whether the error handling interrupt and associated controls are implemented by the node.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UI	Meaning
0b00	Does not support the error handling interrupt. ERR<n>CTLR.UI is RES0.
0b01	Error handling interrupt is supported and always enabled. ERR<n>CTLR.UI is RES0.
0b10	Error handling interrupt is supported and controllable using ERR<n>CTLR.UI.
0b11	Error handling interrupt is supported, and controllable using ERR<n>CTLR.WUI for writes and ERR<n>CTLR.RUI for reads.

Access to this field is **RO**.

IMPLEMENTATION DEFINED, bits [3:2]

IMPLEMENTATION DEFINED.

ED, bits [1:0]

Error reporting and logging. Indicates error record <n> is a normal record and the first record owned the node, and whether the node implements the controls for enabling and disabling error reporting and logging.

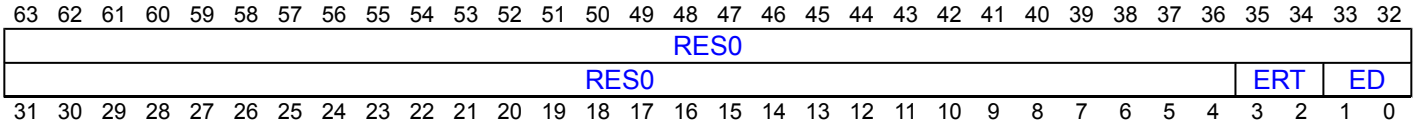
The value of this field is an IMPLEMENTATION DEFINED choice of:

ED	Meaning
0b01	Error reporting and logging always enabled. ERR<n>CTLR.ED is RES0.
0b10	Error reporting and logging is controllable using ERR<n>CTLR.ED.

All other values are reserved.

Access to this field is **RO**.

When RAS System Architecture v2 is implemented and error record <n> is a proxy for a RAS agent:



Bits [63:4]

Reserved, RES0.

ERT, bits [3:2]

Error Record Type. Defines the type of error record.

ERT	Meaning
0b01	Error record is a proxy for a RAS agent.

All other values are reserved.

Access to this field is **RO**.

ED, bits [1:0]

Error reporting and logging. Indicates error record <n> is not a true error record.

ED	Meaning
0b11	Error record <n> is not an error record.

Access to this field is **RO**.

Accessing ERR<n>FR

ERR<n>FR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0x000` + (64 * n)	ERR<n>FR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>MISC0, Error Record <n> Miscellaneous Register 0, n = 0 - 65534

The ERR<n>MISC0 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

If the node that owns error record <n> implements a standard format Corrected error counter or counters (ERRFR[FirstRecordOfNode(n)].CEC != `0b000`), then it is IMPLEMENTATION DEFINED whether error record <n> can record countable errors, and:

- If error record <n> records countable errors, then ERR<n>MISC0 implements the standard format Corrected error counter or counters for error record <n>.
- If error record <n> does not record countable errors, then it is recommended that the fields in ERR<n>MISC0 defined for the standard format counter or counters are RES0. That is, the fields behave like counters that never count.

Configuration

This register is present only when FEAT_RAS is implemented and error record n is implemented. Otherwise, direct accesses to ERR<n>MISC0 are RES0.

ERRFR[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record then FirstRecordOfNode(n) = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC0, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, nonzero, and ignore writes are compliant with this requirement.

Note

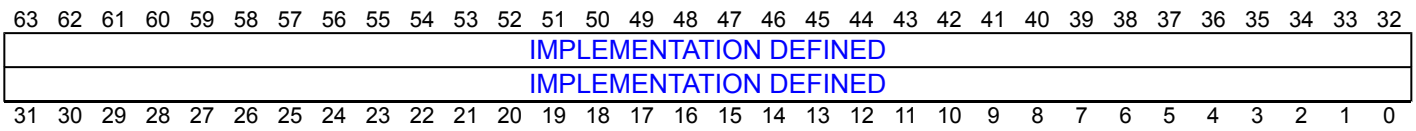
Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in ERRCTLR[FirstRecordOfNode(n)].

Attributes

ERR<n>MISC0 is a 64-bit register.

Field descriptions

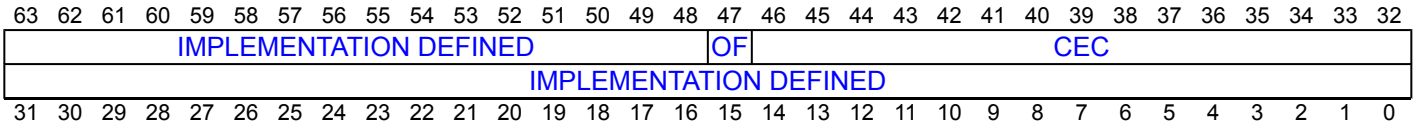
When ERRFR[FirstRecordOfNode(n)].CEC == 0b000 or error record n does not record countable errors:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

When ERRFR[FirstRecordOfNode(n)].CEC == 0b100, ERRFR[FirstRecordOfNode(n)].RP == 0, and error record n records countable errors:



IMPLEMENTATION DEFINED, bits [63:48]

IMPLEMENTATION DEFINED syndrome.

OF, bit [47]

Sticky overflow bit. Set to 1 when ERR<n>MISC0.CEC is incremented and wraps through zero.

OF	Meaning
0b0	Counter has not overflowed.
0b1	Counter has overflowed.

A direct write that modifies this field might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CEC, bits [46:32]

Corrected error count. Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

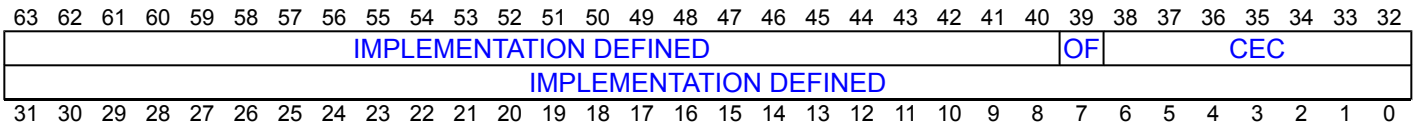
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

When ERRFR[FirstRecordOfNode(n)].CEC == 0b010, ERRFR[FirstRecordOfNode(n)].RP == 0, and error record n records countable errors:



IMPLEMENTATION DEFINED, bits [63:40]

IMPLEMENTATION DEFINED syndrome.

OF, bit [39]

Sticky overflow bit. Set to 1 when ERR<n>MISC0.CEC is incremented and wraps through zero.

OF	Meaning
0b0	Counter has not overflowed.
0b1	Counter has overflowed.

A direct write that modifies this field might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CEC, bits [38:32]

Corrected error count. Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

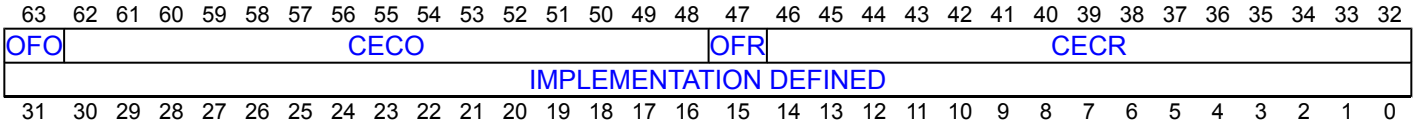
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

When ERRFR[FirstRecordOfNode(n)].CEC == 0b100, ERRFR[FirstRecordOfNode(n)].RP == 1, and error record n records countable errors:



OFO, bit [63]

Sticky overflow bit, other. Set to 1 when ERR<n>MISC0.CECO is incremented and wraps through zero.

OFO	Meaning
0b0	Other counter has not overflowed.
0b1	Other counter has overflowed.

A direct write that modifies this field might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECO, bits [62:48]

Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing ERR<n>MISC0.CECR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

OFR, bit [47]

Sticky overflow bit, repeat. Set to 1 when ERR<n>MISC0.CECR is incremented and wraps through zero.

OFR	Meaning
0b0	Repeat counter has not overflowed.
0b1	Repeat counter has overflowed.

A direct write that modifies this field might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECR, bits [46:32]

Corrected error count, repeat. Incremented for the first countable error, which also records other syndrome for the error, and subsequently for each countable error that matches the recorded other syndrome. Corrected errors are countable errors. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are countable errors.

Note

For example, the other syndrome might include the set and way information for an error detected in a cache. This might be recorded in the IMPLEMENTATION DEFINED ERR<n>MISC<m> fields on a first Corrected error. ERR<n>MISC0.CECR is then incremented for each subsequent Corrected Error in the same set and way.

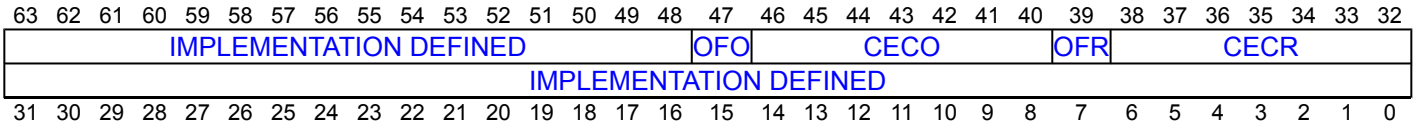
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

When ERRFR[FirstRecordOfNode(n)].CEC == 0b010, ERRFR[FirstRecordOfNode(n)].RP == 1, and error record n records countable errors:



IMPLEMENTATION DEFINED, bits [63:48]

IMPLEMENTATION DEFINED syndrome.

OFO, bit [47]

Sticky overflow bit, other. Set to 1 when ERR<n>MISC0.CECO is incremented and wraps through zero.

OFO	Meaning
0b0	Other counter has not overflowed.
0b1	Other counter has overflowed.

A direct write that modifies this field might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECO, bits [46:40]

Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing ERR<n>MISC0.CECR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

OFR, bit [39]

Sticky overflow bit, repeat. Set to 1 when ERR<n>MISC0.CECR is incremented and wraps through zero.

OFR	Meaning
0b0	Repeat counter has not overflowed.
0b1	Repeat counter has overflowed.

A direct write that modifies this field might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECR, bits [38:32]

Corrected error count, repeat. Incremented for the first countable error, which also records other syndrome for the error, and subsequently for each countable error that matches the recorded other syndrome. Corrected errors are countable errors. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are countable errors.

Note

For example, the other syndrome might include the set and way information for an error detected in a cache. This might be recorded in the IMPLEMENTATION DEFINED ERR<n>MISC<m> fields on a first Corrected error. ERR<n>MISC0.CECR is then incremented for each subsequent Corrected Error in the same set and way.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC0

Reads from ERR<n>MISC0 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and ERRPFGF[FirstRecordOfNode(n)].MV is 1, then some parts of this register are read/write when ERR<n>STATUS.MV is 0. See ERR<n>PFGF.MV for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When ERR<n>STATUS.MV is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC0 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0x020 + (64 * n)	ERR<n>MISC0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>MISC1, Error Record <n> Miscellaneous Register 1, n = 0 - 65534

The ERR<n>MISC1 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

Configuration

This register is present only when FEAT_RAS is implemented and error record n is implemented. Otherwise, direct accesses to ERR<n>MISC1 are RES0.

ERRFR[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record then FirstRecordOfNode(n) = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC1, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, nonzero, and ignore writes are compliant with this requirement.

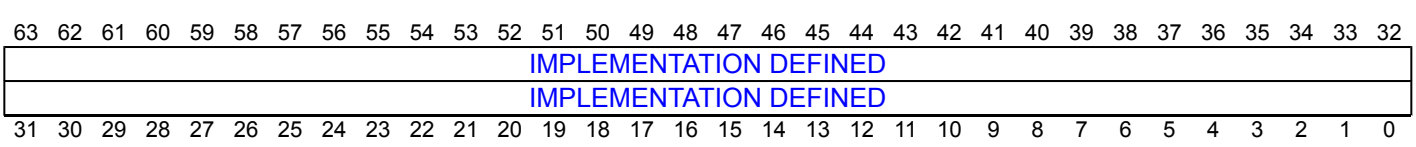
Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in ERRCTLR[FirstRecordOfNode(n)].

Attributes

ERR<n>MISC1 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC1

Reads from ERR<n>MISC1 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and ERRPFGF[FirstRecordOfNode(n)].MV is 1, then some parts of this register are read/write when ERR<n>STATUS.MV is 0. See ERR<n>PFGF.MV for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When ERR<n>STATUS.MV is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC1 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0x028` + (64 * n)	ERR<n>MISC1

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>MISC2, Error Record <n> Miscellaneous Register 2, n = 0 - 65534

The ERR<n>MISC2 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

Configuration

This register is present only when FEAT_RAS is implemented and ((an implementation implements ERR<n>MISC2 or RAS System Architecture v1p1 is implemented) and error record n is implemented). Otherwise, direct accesses to ERR<n>MISC2 are RES0.

ERRFR[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record then FirstRecordOfNode(n) = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC2, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, nonzero, and ignore writes are compliant with this requirement.

Arm recommends that if RAS System Architecture v1.1 is not implemented then ERR<n>MISC2 does not require zeroing to return the record to a quiescent state.

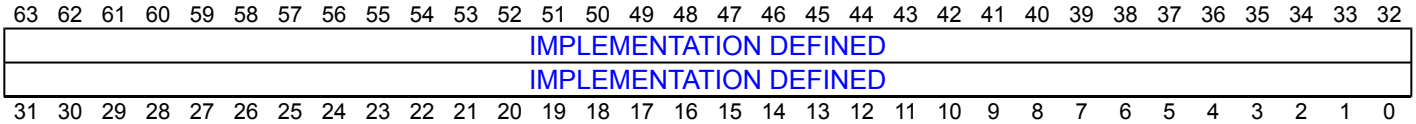
Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in ERRCTLR[FirstRecordOfNode(n)].

Attributes

ERR<n>MISC2 is a 64-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC2

Reads from ERR<n>MISC2 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and ERRPFGF[FirstRecordOfNode(n)].MV is 1, then some parts of this register are read/write when ERR<n>STATUS.MV is 0. See ERR<n>PFGF.MV for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When ERR<n>STATUS.MV is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC2 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0x030` + (64 * n)	ERR<n>MISC2

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>MISC3, Error Record <n> Miscellaneous Register 3, n = 0 - 65534

The ERR<n>MISC3 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

If the node that owns error record n supports the RAS Timestamp Extension (ERRFR[FirstRecordOfNode(n)].TS != 0b00), then ERR<n>MISC3 contains the timestamp value for error record n when the error was detected. Otherwise the contents of ERR<n>MISC3 are IMPLEMENTATION DEFINED.

Configuration

This register is present only when FEAT_RAS is implemented and ((an implementation implements ERR<n>MISC3 or RAS System Architecture v1p1 is implemented) and error record n is implemented). Otherwise, direct accesses to ERR<n>MISC3 are RES0.

ERRFR[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record then FirstRecordOfNode(n) = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC3, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, nonzero, and ignore writes are compliant with this requirement.

Arm recommends that if RAS System Architecture v1.1 is not implemented then ERR<n>MISC3 does not require zeroing to return the record to a quiescent state.

Note

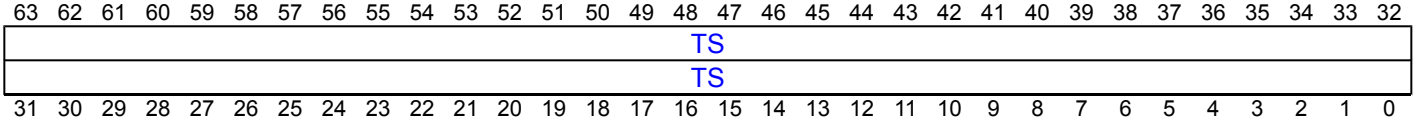
Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in ERRCTLR[FirstRecordOfNode(n)].

Attributes

ERR<n>MISC3 is a 64-bit register.

Field descriptions

When ERRFR[FirstRecordOfNode(n)].TS != 0b00:



TS, bits [63:0]

Timestamp. Timestamp value recorded when the error was detected. Valid only if ERR<n>STATUS.V == 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO or RW**.

When ERRFR[FirstRecordOfNode(n)].TS == 0b00:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
IMPLEMENTATION DEFINED																															
IMPLEMENTATION DEFINED																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC3

Reads from ERR<n>MISC3 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and ERRPFGF[FirstRecordOfNode(n)].MV is 1, then some parts of this register are read/write when ERR<n>STATUS.MV is 0. See ERR<n>PFGF.MV for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When ERR<n>STATUS.MV is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC3 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0x038` + (64 * n)	ERR<n>MISC3

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>PFGCDN, Error Record <n> Pseudo-fault Generation Countdown Register, n = 0 - 65534

The ERR<n>PFGCDN characteristics are:

Purpose

Generates one of the errors enabled in the corresponding ERR<n>PFGCTL register.

Configuration

This register is present only when FEAT_RAS is implemented, error record n is implemented, the node that owns error record n implements the Common Fault Injection Model Extension, and error record n is the first error record in the node. Otherwise, direct accesses to ERR<n>PFGCDN are RES0.

ERR<n>FR describes the features implemented by the node.

Attributes

ERR<n>PFGCDN is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
CDN																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:32]

Reserved, RES0.

CDN, bits [31:0]

Countdown value.

This field is copied to Error Generation Counter when either:

- Software writes 1 to ERR<n>PFGCTL.CDNEN.
- The Error Generation Counter decrements to zero and ERR<n>PFGCTL.R is 1.

While ERR<n>PFGCTL.CDNEN is 1 and the Error Generation Counter is nonzero, the counter decrements by 1 for each cycle at an IMPLEMENTATION DEFINED clock rate. When the counter reaches zero, one of the errors enabled in the ERR<n>PFGCTL register is generated.

Note

The current Error Generation Counter value is not visible to software.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing ERR<n>PFGCDN

This section shows the offset of ERR<n>PFGCDN in an error record group when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, or ERR<n>PFGCDN is accessed in a fault injection group, see 'RAS memory-mapped register views' for the offset of ERR<n>PFGCDN.

ERR<n>PFGCDN can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0x810` + (64 * n)	ERR<n>PFGCDN

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>PFGCTL, Error Record <n> Pseudo-fault Generation Control Register, n = 0 - 65534

The ERR<n>PFGCTL characteristics are:

Purpose

Enables controlled fault generation.

Configuration

This register is present only when FEAT_RAS is implemented, error record n is implemented, the node that owns error record n implements the Common Fault Injection Model Extension, and error record n is the first error record in the node. Otherwise, direct accesses to ERR<n>PFGCTL are RES0.

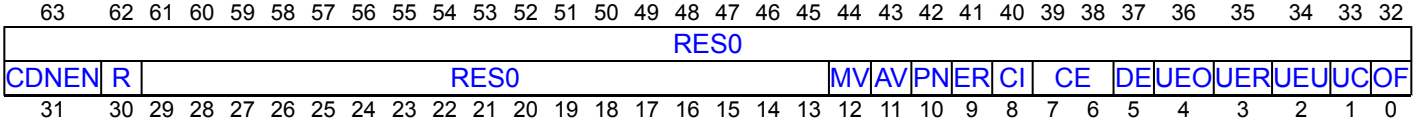
ERR<n>PFGF describes the Common Fault Injection features implemented by the node.

ERR<n>FR describes the features implemented by the node.

Attributes

ERR<n>PFGCTL is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

CDNEN, bit [31]

Countdown Enable. Controls transfers of the value held in ERR<n>PFGCDN to the Error Generation Counter and enables this counter.

CDNEN	Meaning
0b0	The Error Generation Counter is disabled.
0b1	The Error Generation Counter is enabled. On a write of 1 to this field, the Error Generation Counter is set to ERR<n>PFGCDN.CDN.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

R, bit [30]

When ERR<n>PFGF.R == 1:

Restart. Controls whether the Error Generation Counter restarts or stops counting on reaching zero.

R	Meaning
0b0	On reaching zero, the Error Generation Counter will stop counting.
0b1	On reaching zero, the Error Generation Counter is set to ERR<n>PFGCDN.CDN.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:13]

Reserved, RES0.

MV, bit [12]

When ERR<n>PFGF.MV == 1:

Miscellaneous syndrome. The value written to ERR<n>STATUS.MV when an injected error is recorded.

MV	Meaning
0b0	ERR<n>STATUS.MV is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.MV is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the node always sets ERR<n>STATUS.MV to 1 when an injected error is recorded, access to this field is **RAO/WI**.

When the node always sets ERR<n>STATUS.MV to 1 when an injected error is recorded and this field is RAO/WI:

Reserved, RAO/WI.

Otherwise:

Reserved, RES0.

AV, bit [11]

When ERR<n>PFGF.AV == 1:

Address syndrome. The value written to ERR<n>STATUS.AV when an injected error is recorded.

AV	Meaning
0b0	ERR<n>STATUS.AV is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.AV is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the node always sets ERR<n>STATUS.AV to 1 when an injected error is recorded, access to this field is **RAO/WI**.

When the node always sets ERR<n>STATUS.AV to 1 when an injected error is recorded and this field is RAO/WI:

Reserved, RAO/WI.

Otherwise:

Reserved, RES0.

PN, bit [10]
When ERR<n>PFGF.PN == 1:

Poison flag. The value written to ERR<n>STATUS.PN when an injected error is recorded.

PN	Meaning
0b0	ERR<n>STATUS.PN is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.PN is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ER, bit [9]
When ERR<n>PFGF.ER == 1:

Error Reported flag. The value written to ERR<n>STATUS.ER when an injected error is recorded.

ER	Meaning
0b0	ERR<n>STATUS.ER is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.ER is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CI, bit [8]
When ERR<n>PFGF.CI == 1:

Critical Error flag. The value written to ERR<n>STATUS.CI when an injected error is recorded.

CI	Meaning
0b0	ERR<n>STATUS.CI is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.CI is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CE, bits [7:6]

When ERR<n>PFGF.CE != 0b00:

Corrected Error generation enable. Controls the type of injected Corrected error generated by the fault injection feature of the node.

CE	Meaning	Applies when
0b00	An injected Corrected error will not be generated by the fault injection feature of the node.	
0b01	An injected non-specific Corrected error is generated in the fault injection state. ERR<n>STATUS.CE is set to 0b10 when the injected error is recorded.	When ERR<n>PFGF.CE == 0b01
0b10	An injected transient Corrected error is generated in the fault injection state. ERR<n>STATUS.CE is set to 0b01 when the injected error is recorded.	When ERR<n>PFGF.CE == 0b11
0b11	An injected persistent Corrected error is generated in the fault injection state. ERR<n>STATUS.CE is set to 0b11 when the injected error is recorded.	When ERR<n>PFGF.CE == 0b11

The set of permitted values for this field is defined by ERR<n>PFGF.CE.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DE, bit [5]

When ERR<n>PFGF.DE == 1:

Deferred Error generation enable. Controls whether an injected Deferred error is generated by the fault injection feature of the node.

DE	Meaning
0b0	An injected Deferred error will not be generated by the fault generation feature of the node.
0b1	An injected Deferred error is generated in the fault injection state.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UEO, bit [4]

When ERR<n>PFGF.UEO == 1:

Latent or Restartable Error generation enable. Controls whether an injected Latent or Restartable error is generated by the fault injection feature of the node.

UEO	Meaning
0b0	An injected Latent or Restartable error will not be generated by the fault generation feature of the node.
0b1	An injected Latent or Restartable error is generated in the fault injection state.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UER, bit [3]

When ERR<n>PFGF.UER == 1:

Signaled or Recoverable Error generation enable. Controls whether an injected Signaled or Recoverable error is generated by the fault injection feature of the node.

UER	Meaning
0b0	An injected Signaled or Recoverable error will not be generated by the fault generation feature of the node.
0b1	An injected Signaled or Recoverable error is generated in the fault injection state.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UEU, bit [2]

When ERR<n>PFGF.UEU == 1:

Unrecoverable Error generation enable. Controls whether an injected Unrecoverable error is generated by the fault injection feature of the node.

UEU	Meaning
0b0	An injected Unrecoverable error will not be generated by the fault generation feature of the node.
0b1	An injected Unrecoverable error is generated in the fault injection state.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UC, bit [1]

When ERR<n>PFGF.UC == 1:

Uncontainable Error generation enable. Controls whether an injected Uncontainable error is generated by the fault injection feature of the node.

UC	Meaning
0b0	An injected Uncontainable error will not be generated by the fault generation feature of the node.
0b1	An injected Uncontainable error is generated in the fault injection state.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

OF, bit [0]
When ERR<n>PFGF.OF == 1:

Overflow flag. The value written to ERR<n>STATUS.OF when an injected error is recorded.

OF	Meaning
0b0	ERR<n>STATUS.OF is set to 0 when an injected error is recorded.
0b1	ERR<n>STATUS.OF is set to 1 when an injected error is recorded.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing ERR<n>PFGCTL

This section shows the offset of ERR<n>PFGCTL in an error record group when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, or ERR<n>PFGCTL is accessed in a fault injection group, see 'RAS memory-mapped register views' for the offset of ERR<n>PFGCTL.

ERR<n>PFGCTL can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0x808 + (64 * n)	ERR<n>PFGCTL

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>PFGF, Error Record <n> Pseudo-fault Generation Feature Register, n = 0 - 65534

The ERR<n>PFGF characteristics are:

Purpose

Defines which common architecturally-defined fault generation features are implemented.

Configuration

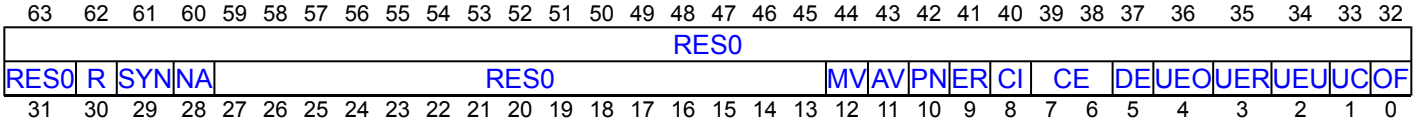
This register is present only when FEAT_RAS is implemented, error record n is implemented, the node that owns error record n implements the Common Fault Injection Model Extension, and error record n is the first error record in the node. Otherwise, direct accesses to ERR<n>PFGF are RES0 .

ERR<n>FR describes the features implemented by the node.

Attributes

ERR<n>PFGF is a 64-bit register.

Field descriptions



Bits [63:31]

Reserved, RES0.

R, bit [30]

Restartable. Support for Error Generation Counter restart mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

R	Meaning
0b0	The node does not support this feature. ERR<n>PFGCTL.R is RES0.
0b1	Error Generation Counter restart mode is implemented and is controlled by ERR<n>PFGCTL.R. ERR<n>PFGCTL.R is a read/write field.

Access to this field is **RO**.

SYN, bit [29]

Syndrome. Fault syndrome injection.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYN	Meaning
0b0	When an injected error is recorded, the node sets ERR<n>STATUS.{IERR, SERR} to IMPLEMENTATION DEFINED values. ERR<n>STATUS.{IERR, SERR} are UNKNOWN when ERR<n>STATUS.V is 0.
0b1	When an injected error is recorded, the node does not update the ERR<n>STATUS.{IERR, SERR} fields. ERR<n>STATUS.{IERR, SERR} are writable when ERR<n>STATUS.V is 0.

Note

If ERR<n>PFGF.SYN is 1 then software can write specific values into the ERR<n>STATUS.{IERR, SERR} fields when setting up a fault injection event. The sets of values that can be written to these fields is IMPLEMENTATION DEFINED.

Access to this field is **RO**.

NA, bit [28]

No access required. Defines whether this component fakes detection of the error on an access to the component or spontaneously in the fault injection state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NA	Meaning
0b0	The component fakes detection of the error on an access to the component.
0b1	The component fakes detection of the error spontaneously in the fault injection state.

Access to this field is **RO**.

Bits [27:13]

Reserved, RES0.

MV, bit [12]

Miscellaneous syndrome.

Defines whether software can control all or part of the syndrome recorded in the ERR<n>MISC<m> registers when an injected error is recorded.

It is IMPLEMENTATION DEFINED which ERR<n>MISC<m> syndrome fields, if any, are updated by the node when an injected error is recorded. Some syndrome fields might always be updated by the node when an error, including an injected error, is recorded. For example, a corrected error counter might always be updated when any countable error, including a injected countable error, is recorded.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MV	Meaning
0b0	When an injected error is recorded, the node might update the ERR<n>MISC<m> registers: • If any syndrome is recorded by the node in the ERR<n>MISC<m> registers, then ERR<n>STATUS.MV is set to 1. • Otherwise, ERR<n>STATUS.MV is unchanged. If the node always sets ERR<n>STATUS.MV to 1 when recording an injected error then ERR<n>PFGCTL.MV might be RAO/WI. Otherwise ERR<n>PFGCTL.MV is RES0.
0b1	When an injected error is recorded, the node might update some, but not all ERR<n>MISC<m> syndrome fields: • If any syndrome is recorded by the node in the ERR<n>MISC<m> registers, then ERR<n>STATUS.MV is set to 1. • Otherwise, ERR<n>STATUS.MV is set to ERR<n>PFGCTL.MV. ERR<n>MISC<m> syndrome fields that are not updated by the node are writable when ERR<n>STATUS.MV is 0. If the node always sets ERR<n>STATUS.MV to 1 when recording an injected error then ERR<n>PFGCTL.MV is RAO/WI. Otherwise ERR<n>PFGCTL.MV is a read/write field.

If ERR<n>PFGF.MV is 1, software can write specific additional syndrome values into the ERR<n>MISC<m> registers when setting up a fault injection event. The permitted values that can be written to these registers are IMPLEMENTATION DEFINED.

Access to this field is **RO**.

AV, bit [11]

Address syndrome. Defines whether software can control the address recorded in ERR<n>ADDR when an injected error is recorded.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AV	Meaning
0b0	When an injected error is recorded, the node might record an address in ERR<n>ADDR. If an address is recorded in ERR<n>ADDR, then ERR<n>STATUS.AV is set to 1. Otherwise, ERR<n>ADDR and ERR<n>STATUS.AV are unchanged. If the node always records an address and sets ERR<n>STATUS.AV to 1 when recording an injected error then ERR<n>PFGCTL.AV might be RAO/WI. Otherwise ERR<n>PFGCTL.AV is RES0.
0b1	When an injected error is recorded, the node does not update ERR<n>ADDR and does one of: • Sets ERR<n>STATUS.AV to ERR<n>PFGCTL.AV. ERR<n>PFGCTL.AV is a read/write field. • Sets ERR<n>STATUS.AV to 1. ERR<n>PFGCTL.AV is RAO/WI. ERR<n>ADDR is writable when ERR<n>STATUS.AV is 0.

If ERR<n>PFGF.AV is 1 then software can write a specific address value into ERR<n>ADDR when setting up a fault injection event.

Access to this field is **RO**.

PN, bit [10]

When the node supports this flag:

Poison flag. Describes how the fault generation feature of the node sets the ERR<n>STATUS.PN status flag.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PN	Meaning
0b0	When an injected error is recorded, it is IMPLEMENTATION DEFINED whether the node sets ERR<n>STATUS.PN to 1. ERR<n>PFGCTL.PN is RES0.
0b1	When an injected error is recorded, ERR<n>STATUS.PN is set to ERR<n>PFGCTL.PN. ERR<n>PFGCTL.PN is a read/write field.

This behavior replaces the architecture-defined rules for setting the ERR<n>STATUS.PN bit.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

ER, bit [9]

When the node supports this flag:

Error Reported flag. Describes how the fault generation feature of the node sets the ERR<n>STATUS.ER status flag.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ER	Meaning
0b0	When an injected error is recorded, the node sets ERR<n>STATUS.ER according to the architecture-defined rules for setting the ER field. ERR<n>PFGCTL.ER is RES0.
0b1	When an injected error is recorded, ERR<n>STATUS.ER is set to ERR<n>PFGCTL.ER. This behavior replaces the architecture-defined rules for setting the ER bit. ERR<n>PFGCTL.ER is a read/write field.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

CI, bit [8]

When the node supports this flag:

Critical Error flag. Describes how the fault generation feature of the node sets the ERR<n>STATUS.CI status flag.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CI	Meaning
0b0	When an injected error is recorded, it is IMPLEMENTATION DEFINED whether the node sets ERR<n>STATUS.CI to 1. ERR<n>PFGCTL.CI is RES0.
0b1	When an injected error is recorded, ERR<n>STATUS.CI is set to ERR<n>PFGCTL.CI. ERR<n>PFGCTL.CI is a read/write field.

This behavior replaces the architecture-defined rules for setting the ERR<n>STATUS.CI bit.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

CE, bits [7:6]

When the node supports this type of error:

Corrected Error generation. Describes the types of Corrected error that the fault generation feature of the node can generate.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CE	Meaning
0b00	The fault generation feature of the node does not generate Corrected errors. ERR<n>PFGCTL.CE is RES0.
0b01	The fault generation feature of the node allows generation of a non-specific Corrected error, that is, a Corrected error that is recorded by setting ERR<n>STATUS.CE to 0b10. ERR<n>PFGCTL.CE is a read/write field. The values 0b10 and 0b11 in ERR<n>PFGCTL.CE are reserved.
0b11	The fault generation feature of the node allows generation of transient or persistent Corrected errors, that is, Corrected errors that are recorded by setting ERR<n>STATUS.CE to 0b01 or 0b11 respectively. ERR<n>PFGCTL.CE is a read/write field. The value 0b01 in ERR<n>PFGCTL.CE is reserved.

All other values are reserved.

If ERR<n>FR.FRX is 1 then ERR<n>FR.CE indicates whether the node supports this type of error.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

DE, bit [5]

When the node supports this type of error:

Deferred Error generation. Describes whether the fault generation feature of the node can generate Deferred errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DE	Meaning
0b0	The fault generation feature of the node does not generate Deferred errors. ERR<n>PFGCTL.DE is RES0.
0b1	The fault generation feature of the node allows generation of Deferred errors. ERR<n>PFGCTL.DE is a read/write field.

If ERR<n>FR.FRX is 1 then ERR<n>FR.DE indicates whether the node supports this type of error.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

UEO, bit [4]

When the node supports this type of error:

Latent or Restartable Error generation. Describes whether the fault generation feature of the node can generate Latent or Restartable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEO	Meaning
0b0	The fault generation feature of the node does not generate Latent or Restartable errors. ERR<n>PFGCTL.UEO is RES0.
0b1	The fault generation feature of the node allows generation of Latent or Restartable errors. ERR<n>PFGCTL.UEO is a read/write field.

If ERR<n>FR.FRX is 1 then ERR<n>FR.UEO indicates whether the node supports this type of error.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

UER, bit [3]

When the node supports this type of error:

Signaled or Recoverable Error generation. Describes whether the fault generation feature of the node can generate Signaled or Recoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UER	Meaning
0b0	The fault generation feature of the node does not generate Signaled or Recoverable errors. ERR<n>PFGCTL.UER is RES0.
0b1	The fault generation feature of the node allows generation of Signaled or Recoverable errors. ERR<n>PFGCTL.UER is a read/write field.

If ERR<n>FR.FRX is 1 then ERR<n>FR.UER indicates whether the node supports this type of error.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

UEU, bit [2]

When the node supports this type of error:

Unrecoverable Error generation. Describes whether the fault generation feature of the node can generate Unrecoverable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UEU	Meaning
0b0	The fault generation feature of the node does not generate Unrecoverable errors. ERR<n>PFGCTL.UEU is RES0.
0b1	The fault generation feature of the node allows generation of Unrecoverable errors. ERR<n>PFGCTL.UEU is a read/write field.

If ERR<n>FR.FRX is 1 then ERR<n>FR.UEU indicates whether the node supports this type of error.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

UC, bit [1]

When the node supports this type of error:

Uncontainable Error generation. Describes whether the fault generation feature of the node can generate Uncontainable errors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UC	Meaning
0b0	The fault generation feature of the node does not generate Uncontainable errors. ERR<n>PFGCTL.UC is RES0.
0b1	The fault generation feature of the node allows generation of Uncontainable errors. ERR<n>PFGCTL.UC is a read/write field.

If ERR<n>FR.FRX is 1 then ERR<n>FR.UC indicates whether the node supports this type of error.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

OF, bit [0]

When the node supports this flag:

Overflow flag. Describes how the fault generation feature of the node sets the ERR<n>STATUS.OF status flag.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OF	Meaning
0b0	When an injected error is recorded, the node sets ERR<n>STATUS.OF according to the architecture-defined rules for setting the OF field. ERR<n>PFGCTL.OF is RES0.
0b1	When an injected error is recorded, ERR<n>STATUS.OF is set to ERR<n>PFGCTL.OF. This behavior replaces the architecture-defined rules for setting the OF bit. ERR<n>PFGCTL.OF is a read/write field.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

Accessing ERR<n>PFGF

This section shows the offset of ERR<n>PFGF in an error record group when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, or ERR<n>PFGF is accessed in a fault injection group, see 'RAS memory-mapped register views' for the offset of ERR<n>PFGF.

ERR<n>PFGF can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0x800 + (64 * n)	ERR<n>PFGF

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>STATUS, Error Record <n> Primary Status Register, n = 0 - 65534

The ERR<n>STATUS characteristics are:

Purpose

When RAS System Architecture v2 is implemented, error record <n> might be one of the following:

- A continuation record containing more information about the error recorded in error record <n-1>. In this case, ERR<n>STATUS contains a subset of the values of a normal error record status register.
- A proxy for a different RAS agent. In this case, ERR<n>STATUS reports the status of the RAS agent.

Otherwise, ERR<n>STATUS contains status information for error record <n>, including:

- Whether any error has been detected (valid).
- Whether any detected error was not corrected, and returned to a Requester.
- Whether any detected error was not corrected and deferred.
- Whether an error record has been discarded because additional errors have been detected before the first error was handled by software (overflow).
- Whether any error has been reported.
- Whether the other error record registers contain valid information.
- Whether the error was reported because poison data was detected or because a corrupt value was detected by an error detection code.
- A primary error code.
- An IMPLEMENTATION DEFINED extended error code.

Within this register:

- ERR<n>STATUS.{AV, V, MV} are valid bits that define whether error record <n> registers are valid.
- ERR<n>STATUS.{UE, OF, CE, DE, UET} encode the types of error or errors recorded.
- ERR<n>STATUS.{CI, ER, PN, IERR, SERR} are syndrome fields.

Configuration

This register is present only when FEAT_RAS is implemented and error record n is implemented. Otherwise, direct accesses to ERR<n>STATUS are RES0.

ERRFRPFGF[FirstRecordOfNode(n)] describes the features implemented by the node that owns error record <n>. FirstRecordOfNode(n) is the index of the first error record owned by the same node as error record <n>. If the node owns a single record then FirstRecordOfNode(n) = n.

For IMPLEMENTATION DEFINED fields in ERR<n>STATUS, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, nonzero, and ignore writes are compliant with this requirement.

Note

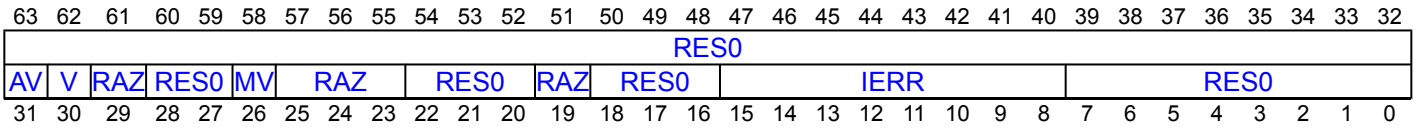
Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in ERRCTLR[FirstRecordOfNode(n)].

Attributes

ERR<n>STATUS is a 64-bit register.

Field descriptions

When RAS System Architecture v2 is implemented, ERR<n>FR.ED == 0b00, and ERR<n>FR.ERT == 0b01:



Continuation record.

Bits [63:32]

Reserved, RES0.

AV, bit [31]

When error record n includes an address associated with an error:

Address Valid.

AV	Meaning
0b0	ERR<n>ADDR not valid.
0b1	ERR<n>ADDR contains an additional address associated with the highest priority error recorded by this record.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == 1, this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

V, bit [30]

Status Register Valid.

V	Meaning
0b0	ERR<n>STATUS not valid.
0b1	ERR<n>STATUS valid. Additional syndrome has been recorded.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == 1, this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Access to this field is **W1C**.

Bit [29]

Reserved, RAZ.

Bits [28:27]

Reserved, RES0.

MV, bit [26]

When error record <n> includes additional information for an error:

Miscellaneous Registers Valid.

MV	Meaning
0b0	ERR<n>MISC<m> not valid.
0b1	The contents of the ERR<n>MISC<m> registers contain additional information for an error recorded by this record.

Note

If the ERR<n>MISC<m> registers can contain additional information for a previously recorded error, then the contents must be self-describing to software or a user. For example, certain fields might relate only to Corrected errors, and other fields only to the most recent error that was not discarded.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == 1, this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

Bits [25:23]

Reserved, RAZ.

Bits [22:20]

Reserved, RES0.

Bit [19]

Reserved, RAZ.

Bits [18:16]

Reserved, RES0.

IERR, bits [15:8]

IMPLEMENTATION DEFINED additional error code. Used with any primary error code ERR<n>STATUS.SERR value. Further IMPLEMENTATION DEFINED information can be placed in the ERR<n>MISC<m> registers.

The implemented set of valid values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

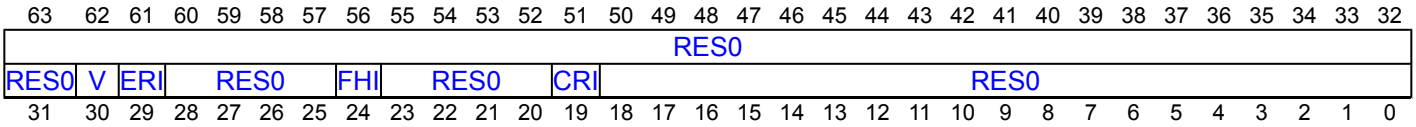
Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
  - ERR<n>STATUS.V == 0.
  - the node that owns error record n does not implement the Common Fault Injection Model Extension.
- Access to this field is **UNKNOWN/WI** if all the following are true:
  - ERR<n>STATUS.V == 0.
  - ERRPFGF[FirstRecordOfNode(n)].SYN == 0.
- Otherwise, access to this field is **RW**.

Bits [7:0]

Reserved, RES0.

When RAS System Architecture v2 is implemented, ERR<n>FR.ED == 0b11, and ERR<n>FR.ERT == 0b01:



Proxy for a RAS agent.

Bits [63:31]

Reserved, RES0.

V, bit [30]

RAS agent error status.

V	Meaning
0b0	RAS agent error status is not asserted.
0b1	RAS agent error status is asserted.

Access to this field is **RO**.

ERI, bit [29]

RAS agent Error Recovery condition.

ERI	Meaning
0b0	RAS agent error recovery condition is false.
0b1	RAS agent error recovery condition is true.

Access to this field is **RO**.

Bits [28:25]

Reserved, RES0.

FHI, bit [24]

RAS agent Fault Handling condition.

FHI	Meaning
0b0	RAS agent fault handling condition is false.
0b1	RAS agent fault handling condition is true.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

CRI, bit [19]

RAS agent criticial error condition.

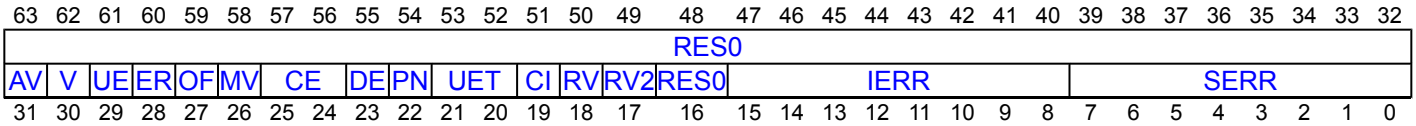
CRI	Meaning
0b0	RAS agent criticial error condition is false.
0b1	RAS agent criticial error condition is true.

Access to this field is **RO**.

Bits [18:0]

Reserved, RES0.

When RAS System Architecture v1p1 is implemented:



Normal record, from FEAT_RASSAv1p1.

Bits [63:32]

Reserved, RES0.

AV, bit [31]

When error record n includes an address associated with an error:

Address Valid.

AV	Meaning
0b0	ERR<n>ADDR not valid.
0b1	ERR<n>ADDR contains an address associated with the highest priority error recorded by this record.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == 1, this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

V, bit [30]

Status Register Valid.

V	Meaning
0b0	ERR<n>STATUS not valid.
0b1	ERR<n>STATUS valid. At least one error has been recorded.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == 1, this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Access to this field is **W1C**.

UE, bit [29]

Uncorrected Error.

UE	Meaning
0b0	No errors have been detected, or all detected errors have been either corrected or deferred.
0b1	At least one detected error was not corrected and not deferred.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

ER, bit [28]

When in-band error responses can be returned for a Deferred error:

Error Reported.

ER	Meaning
0b0	No in-band error response (External abort) signaled to the Requester making the access or other transaction.
0b1	An in-band error response was signaled by the component to the Requester making the access or other transaction. This can be because any of the following are true: • The ERRCTLR[FirstRecordOfNode(n)].UE field, or applicable one of the ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE} fields, is implemented and was 1 when an error was detected and not corrected. • The ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE, UE} fields are not implemented and the component always reports errors.

Note

An in-band error response signaled by the component might be masked and not generate any exception.

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as a Deferred error, but is not deferred to the Requester, can signal an in-band error response to the Requester, causing this field to be set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - ERR<n>STATUS.V == 0.
  - ERR<n>STATUS.[DE,UE] == 0b00.
- Otherwise, access to this field is **W1C**.

When in-band error responses are never returned for a Deferred error:

Error Reported.

ER	Meaning
0b0	No in-band error response (External abort) signaled to the Requester making the access or other transaction.
0b1	An in-band error response was signaled by the component to the Requester making the access or other transaction. This can be because any of the following are true: • The ERRCTLR[FirstRecordOfNode(n)].UE field, or applicable one of the ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE} fields, is implemented and was 1 when an error was detected and not corrected. • The ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE, UE} fields are not implemented and the component always reports errors.

Note

An in-band error response signaled by the component might be masked and not generate any exception.

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as a Deferred error, but is not deferred to the Requester, can signal an in-band error response to the Requester, causing this field to be set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - ERR<n>STATUS.V == 0.
  - ERR<n>STATUS.UE == 0.
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

OF, bit [27]

Overflow.

Indicates that multiple errors have been detected. This field is set to 1 when one of the following occurs:

- A Corrected error counter is implemented, an error is counted, and the counter overflows.
- ERR<n>STATUS.V was previously 1, a Corrected error counter is not implemented, and a Corrected error is recorded.
- ERR<n>STATUS.V was previously 1, and a type of error other than a Corrected error is recorded.

Otherwise, this field is unchanged when an error is recorded.

If a Corrected error counter is implemented, then:

- A direct write that modifies the counter overflow flag indirectly might set this field to an UNKNOWN value.
- A direct write to this field that clears this field to zero might indirectly set the counter overflow flag to an UNKNOWN value.

OF	Meaning
0b0	Since this field was last cleared to zero, no error syndrome has been discarded and, if a Corrected error counter is implemented, it has not overflowed.
0b1	Since this field was last cleared to zero, at least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

MV, bit [26]

When error record <n> includes additional information for an error:

Miscellaneous Registers Valid.

MV	Meaning
0b0	ERR<n>MISC<m> not valid.
0b1	The contents of the ERR<n>MISC<m> registers contain additional information for an error recorded by this record.

Note

If the ERR<n>MISC<m> registers can contain additional information for a previously recorded error, then the contents must be self-describing to software or a user. For example, certain fields might relate only to Corrected errors, and other fields only to the most recent error that was not discarded.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == 1, this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

CE, bits [25:24]

Corrected Error.

CE	Meaning
0b00	No errors were corrected.
0b01	At least one transient error was corrected.
0b10	At least one error was corrected.
0b11	At least one persistent error was corrected.

The mechanism by which a component or node detects whether a Corrected error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to 0b10 when a corrected error is recorded.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

DE, bit [23]

Deferred Error.

DE	Meaning
0b0	No errors were deferred.
0b1	At least one error was not corrected and deferred.

Support for deferring errors is IMPLEMENTATION DEFINED.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

PN, bit [22]

Poison.

PN	Meaning
0b0	Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC), or Corrected error recorded.
0b1	Uncorrected error or Deferred error recorded because a poison value was detected.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - ERR<n>STATUS.V == 0.
  - ERR<n>STATUS.[DE,UE] == 0b00.
- Otherwise, access to this field is **W1C**.

UET, bits [21:20]

Uncorrected Error Type. Describes the state of the component after detecting or consuming an Uncorrected error.

UET	Meaning
0b00	Uncorrected error, Uncontainable error (UC).
0b01	Uncorrected error, Unrecoverable error (UEU).
0b10	Uncorrected error, Latent or Restartable error (UEO).
0b11	Uncorrected error, Signaled or Recoverable error (UER).

UER can mean either Signaled or Recoverable error, and UEO can mean either Latent or Restartable error.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - ERR<n>STATUS.V == 0.
  - ERR<n>STATUS.UE == 0.
- Otherwise, access to this field is **W1C**.

CI, bit [19]

Critical Error. Indicates whether a critical error condition has been recorded.

CI	Meaning
0b0	No critical error condition.
0b1	Critical error condition.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

RV, bit [18]

When RAS System Architecture v2 is implemented:

Reset Valid. When ERR<n>STATUS.V is 1, indicating the error record is valid, this field indicates whether the error was recorded before or after the most recent Error Recovery reset.

RV	Meaning
0b0	If the error record is valid then one or more errors have been recorded after the last Error Recovery reset. This error or errors might have overwritten lower priority errors recorded before the last Error Recovery reset.
0b1	If the error record is valid then one or more errors were recorded before the last Error Recovery reset.

This field is set to 0 when an error is recorded and either the fault overwrites the error syndrome, or the error record was previously not valid.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to '1'.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

RV2, bit [17]

When RAS System Architecture v2 is implemented:

Reset Valid 2. When ERR<n>STATUS.{V, RV} is {1, 1}, indicating the error record is valid and one or more errors were recorded before the last Error Recovery reset, this field indicates whether any lower severity errors have been recorded after the Error Recovery reset that did not overwrite the syndrome.

RV2	Meaning
0b0	If the error record is valid then one or more errors were recorded after the last Error Recovery reset that did not overwrite the error syndrome. This includes errors that did not overwrite a previously recorded error syndrome.
0b1	If the error record is valid then one or more errors were recorded before the last Error Recovery reset.

This field is set to 0 when an error is recorded, including when the fault does not overwrite a previously recorded syndrome.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to '1'.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

Bit [16]

Reserved, RES0.

IERR, bits [15:8]

IMPLEMENTATION DEFINED error code. Used with any primary error code ERR<n>STATUS.SERR value. Further IMPLEMENTATION DEFINED information can be placed in the ERR<n>MISC<m> registers.

The implemented set of valid values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
  - the node that owns error record n does not implement the Common Fault Injection Model Extension.
  - ERR<n>STATUS.V == 0.
- Access to this field is **UNKNOWN/WI** if all the following are true:
  - ERRPFGF[FirstRecordOfNode(n)].SYN == 0.
  - ERR<n>STATUS.V == 0.
- Otherwise, access to this field is **RW**.

SERR, bits [7:0]

Architecturally-defined primary error code. The primary error code might be used by a fault handling agent to triage an error without requiring device-specific code. For example, to count and threshold corrected errors in software, or generate a short log entry.

SERR	Meaning
0x00	No error.
0x01	IMPLEMENTATION DEFINED error.
0x02	Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer.
0x03	IMPLEMENTATION DEFINED pin. For example, **nSEI** pin.
0x04	Assertion failure. For example, consistency failure.
0x05	Error detected on internal data path. For example, parity on ALU result.
0x06	Data value from associative memory. For example, ECC error on cache data.
0x07	Address/control value from associative memory. For example, ECC error on cache tag.
0x08	Data value from a TLB. For example, ECC error on TLB data.
0x09	Address/control value from a TLB. For example, ECC error on TLB tag.
0x0A	Data value from producer. For example, parity error on write data bus.
0x0B	Address/control value from producer. For example, parity error on address bus.
0x0C	Data value from (non-associative) external memory. For example, ECC error in SDRAM.
0x0D	Illegal address (software fault). For example, access to unpopulated memory.
0x0E	Illegal access (software fault). For example, byte write to word register.
0x0F	Illegal state (software fault). For example, device not ready.
0x10	Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, SIMD&FP, SVE, and SME registers are data registers.
0x11	Internal control register. For example, parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&FP, SVE, and SME registers are control registers.
0x12	Error response from Completer of access. For example, error response from cache write-back.
0x13	External timeout. For example, timeout on interaction with another component.
0x14	Internal timeout. For example, timeout on interface within the component.
0x15	Deferred error from Completer not supported at Requester. For example, poisoned data received from the Completer of an access by a Requester that cannot defer the error further.
0x16	Deferred error from Requester not supported at Completer. For example, poisoned data received from the Requester of an access by a Completer that cannot defer the error further.
0x17	Deferred error from Completer passed through. For example, poisoned data received from the Completer of an access and returned to the Requester.
0x18	Deferred error from Requester passed through. For example, poisoned data received from the Requester of an access and deferred to the Completer.
0x19	Error recorded by PCIe error logs. Indicates that the component has recorded an error in a PCIe error log. This might be the PCIe device status register, AER, DVSEC, or other mechanisms defined by PCIe.
0x1A	Other internal error. For example, parity error on internal state of the component that is not covered by another primary error code.

All other values are reserved.

The implemented set of valid values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

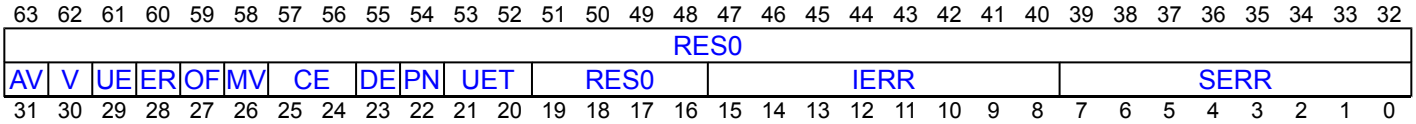
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
  - the node that owns error record n does not implement the Common Fault Injection Model Extension.
  - ERR<n>STATUS.V == 0.
- Access to this field is **UNKNOWN/WI** if all the following are true:
  - ERRPFGF[FirstRecordOfNode(n)].SYN == 0.
  - ERR<n>STATUS.V == 0.

- Otherwise, access to this field is **RW**.

Otherwise:



Normal record, when FEAT_RASSAv1p1 is not implemented.

Bits [63:32]

Reserved, RES0.

AV, bit [31]

When error record n includes an address associated with an error:

Address Valid.

AV	Meaning
0b0	ERR<n>ADDR not valid.
0b1	ERR<n>ADDR contains an address associated with the highest priority error recorded by this record.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == 1, this field resets to an architecturally UNKNOWN value.
  - Otherwise, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.[DE,UE] == 0b00.
  - ERR<n>STATUS.CE != 0b00.
  - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE == 0.
  - ERR<n>STATUS.DE != 0.
  - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE != 0.
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

V, bit [30]

Status Register Valid.

V	Meaning
0b0	ERR<n>STATUS not valid.
0b1	ERR<n>STATUS valid. At least one error has been recorded.

The reset behavior of this field is:

- On a Cold reset:

    - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == 1, this field resets to an architecturally UNKNOWN value.
    - Otherwise, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RO** if all the following are true:
    - ERR<n>STATUS.CE != 0b00.
    - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write.
- Access to this field is **RO** if all the following are true:
    - ERR<n>STATUS.DE != 0.
    - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write.
- Access to this field is **RO** if all the following are true:
    - ERR<n>STATUS.UE != 0.
    - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **W1C**.

UE, bit [29]

Uncorrected Error.

UE	Meaning
0b0	No errors have been detected, or all detected errors have been either corrected or deferred.
0b1	At least one detected error was not corrected and not deferred.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Access to this field is **RO** if all the following are true:
    - ERR<n>STATUS.OF == 1.
    - ERR<n>STATUS.OF is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **W1C**.

ER, bit [28]

When in-band error responses can be returned for a Deferred error:

Error Reported.

ER	Meaning
0b0	No in-band error response (External abort) signaled to the Requester making the access or other transaction.
0b1	An in-band error response was signaled by the component to the Requester making the access or other transaction. This can be because any of the following are true: • The ERRCTLR[FirstRecordOfNode(n)].UE field, or applicable one of the ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE} fields, is implemented and was 1 when an error was detected and not corrected. • The ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE, UE} fields are not implemented and the component always reports errors.

If this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero, when any of:

- Clearing ERR<n>STATUS.V to 0.
- Clearing both ERR<n>STATUS.{DE, UE} to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - ERR<n>STATUS.V == 0.
  - ERR<n>STATUS.[DE,UE] == 0b00.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE != 0.
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE == 0.
  - ERR<n>STATUS.DE != 0.
  - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **W1C**.

When in-band error responses are never returned for a Deferred error:

Error Reported.

ER	Meaning
0b0	No in-band error response (External abort) signaled to the Requester making the access or other transaction.
0b1	An in-band error response was signaled by the component to the Requester making the access or other transaction. This can be because any of the following are true: • The ERRCTLR[FirstRecordOfNode(n)].UE field, or applicable one of the ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE} fields, is implemented and was 1 when an error was detected and not corrected. • The ERRCTLR[FirstRecordOfNode(n)].{WUE, RUE, UE} fields are not implemented and the component always reports errors.

If this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero, when any of:

- Clearing ERR<n>STATUS.V to 0.
- Clearing ERR<n>STATUS.UE to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - ERR<n>STATUS.V == 0.
  - ERR<n>STATUS.UE == 0.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE != 0.
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

OF, bit [27]

Overflow.

Indicates that multiple errors have been detected. This field is set to 1 when one of the following occurs:

- An Uncorrected error is detected and ERR<n>STATUS.UE == 1.
- A Deferred error is detected, ERR<n>STATUS.UE == 0 and ERR<n>STATUS.DE == 1.
- A Corrected error is detected, no Corrected error counter is implemented, ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, and ERR<n>STATUS.CE != 0b00. ERR<n>STATUS.CE might be updated for the new Corrected error.
- A Corrected error counter is implemented, ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this field is set to 1 when one of the following occurs:

- A Deferred error is detected and ERR<n>STATUS.UE == 1.
- A Corrected error is detected, no Corrected error counter is implemented, and ERR<n>STATUS.{UE, DE} != {0, 0}.
- A Corrected error counter is implemented, ERR<n>STATUS.{UE, DE} != {0, 0}, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this field is cleared to 0 when one of the following occurs:

- An Uncorrected error is detected and ERR<n>STATUS.UE == 0.
- A Deferred error is detected, ERR<n>STATUS.UE == 0, and ERR<n>STATUS.DE == 0.
- A Corrected error is detected, ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, and ERR<n>STATUS.CE == `0b00`.

The IMPLEMENTATION DEFINED clearing of this field might also depend on the value of the other error status fields.

If a Corrected error counter is implemented, then:

- A direct write that modifies the counter overflow flag indirectly might set this field to an UNKNOWN value.
- A direct write to this field that clears this field to 0 might indirectly set the counter overflow flag to an UNKNOWN value.

OF	Meaning
`0b0`	If ERR<n>STATUS.UE == 1, then no error syndrome for an Uncorrected error has been discarded. If ERR<n>STATUS.UE == 0 and ERR<n>STATUS.DE == 1, then no error syndrome for a Deferred error has been discarded. If ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, and a Corrected error counter is implemented, then the counter has not overflowed. If ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, ERR<n>STATUS.CE != `0b00`, and no Corrected error counter is implemented, then no error syndrome for a Corrected error has been discarded. **Note** This field might have been set to 1 when an error syndrome was discarded and later cleared to 0 when a higher priority syndrome was recorded.
`0b1`	At least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

MV, bit [26]
When error record <n> includes additional information for an error:

Miscellaneous Registers Valid.

MV	Meaning
`0b0`	ERR<n>MISC<m> not valid.
`0b1`	The contents of the ERR<n>MISC<m> registers contain additional information for an error recorded by this record.

Note

If the ERR<n>MISC<m> registers can contain additional information for a previously recorded error, then the contents must be self-describing to software or a user. For example, certain fields might relate only to Corrected errors, and other fields only to the most recent error that was not discarded.

The reset behavior of this field is:

- On a Cold reset:
  - When RAS System Architecture v2 is implemented and ERRFR[FirstRecordOfNode(n)].SRV == 1, this field resets to an architecturally UNKNOWN value.

- Otherwise, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.[DE,UE] == 0b00.
  - ERR<n>STATUS.CE != 0b00.
  - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE == 0.
  - ERR<n>STATUS.DE != 0.
  - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE != 0.
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

CE, bits [25:24]

Corrected Error.

CE	Meaning
0b00	No errors were corrected.
0b01	At least one transient error was corrected.
0b10	At least one error was corrected.
0b11	At least one persistent error was corrected.

The mechanism by which a component or node detects whether a Corrected error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to 0b10 when a corrected error is recorded.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.OF == 1.
  - ERR<n>STATUS.OF is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **W1C**.

DE, bit [23]

Deferred Error.

DE	Meaning
0b0	No errors were deferred.
0b1	At least one error was not corrected and deferred.

Support for deferring errors is IMPLEMENTATION DEFINED.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.OF == 1.
  - ERR<n>STATUS.OF is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **W1C**.

PN, bit [22]

Poison.

PN	Meaning
0b0	Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC), or Corrected error recorded.
0b1	Uncorrected error or Deferred error recorded because a poison value was detected.

If this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero, when any of:

- Clearing ERR<n>STATUS.V to 0.
- Clearing both ERR<n>STATUS.{DE, UE} to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:
  - ERR<n>STATUS.V == 0.
  - ERR<n>STATUS.[DE,UE] == 0b00.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.[DE,UE] == 0b00.
  - ERR<n>STATUS.CE != 0b00.
  - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE == 0.
  - ERR<n>STATUS.DE != 0.
  - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE != 0.
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **W1C**.

UET, bits [21:20]

Uncorrected Error Type. Describes the state of the component after detecting or consuming an Uncorrected error.

UET	Meaning
0b00	Uncorrected error, Uncontainable error (UC).
0b01	Uncorrected error, Unrecoverable error (UEU).
0b10	Uncorrected error, Latent or Restartable error (UEO).
0b11	Uncorrected error, Signaled or Recoverable error (UER).

UER can mean either Signaled or Recoverable error, and UEO can mean either Latent or Restartable error.

If this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero, when any of:

- Clearing ERR<n>STATUS.V to 0.
- Clearing ERR<n>STATUS.UE to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if any the following are true:

        - ERR<n>STATUS.V == 0.
        - ERR<n>STATUS.UE == 0.
    - Access to this field is **RO** if all the following are true:
        - ERR<n>STATUS.[DE,UE] == 0b00.
        - ERR<n>STATUS.CE != 0b00.
        - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write.
    - Access to this field is **RO** if all the following are true:
        - ERR<n>STATUS.UE == 0.
        - ERR<n>STATUS.DE != 0.
        - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write.
    - Access to this field is **RO** if all the following are true:
        - ERR<n>STATUS.UE != 0.
        - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write.
    - Otherwise, access to this field is **W1C**.

Bits [19:16]

Reserved, RES0.

IERR, bits [15:8]

IMPLEMENTATION DEFINED error code. Used with any primary error code ERR<n>STATUS.SERR value. Further IMPLEMENTATION DEFINED information can be placed in the ERR<n>MISC<m> registers.

The implemented set of valid values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
    - the node that owns error record n does not implement the Common Fault Injection Model Extension.
    - ERR<n>STATUS.V == 0.
- Access to this field is **UNKNOWN/WI** if all the following are true:
    - ERRPFGF[FirstRecordOfNode(n)].SYN == 0.
    - ERR<n>STATUS.V == 0.
- Access to this field is **RO** if all the following are true:
    - ERR<n>STATUS.[DE,UE] == 0b00.
    - ERR<n>STATUS.CE != 0b00.
    - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write.
- Access to this field is **RO** if all the following are true:
    - ERR<n>STATUS.UE == 0.
    - ERR<n>STATUS.DE != 0.
    - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write.
- Access to this field is **RO** if all the following are true:
    - ERR<n>STATUS.UE != 0.
    - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **RW**.

SERR, bits [7:0]

Architecturally-defined primary error code. The primary error code might be used by a fault handling agent to triage an error without requiring device-specific code. For example, to count and threshold corrected errors in software, or generate a short log entry.

SERR	Meaning
0x00	No error.
0x01	IMPLEMENTATION DEFINED error.
0x02	Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer.
0x03	IMPLEMENTATION DEFINED pin. For example, **nSEI** pin.
0x04	Assertion failure. For example, consistency failure.
0x05	Error detected on internal data path. For example, parity on ALU result.
0x06	Data value from associative memory. For example, ECC error on cache data.
0x07	Address/control value from associative memory. For example, ECC error on cache tag.
0x08	Data value from a TLB. For example, ECC error on TLB data.
0x09	Address/control value from a TLB. For example, ECC error on TLB tag.
0x0A	Data value from producer. For example, parity error on write data bus.
0x0B	Address/control value from producer. For example, parity error on address bus.
0x0C	Data value from (non-associative) external memory. For example, ECC error in SDRAM.
0x0D	Illegal address (software fault). For example, access to unpopulated memory.
0x0E	Illegal access (software fault). For example, byte write to word register.
0x0F	Illegal state (software fault). For example, device not ready.
0x10	Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, SIMD&FP, SVE, and SME registers are data registers.
0x11	Internal control register. For example, parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&FP, SVE, and SME registers are control registers.
0x12	Error response from Completer of access. For example, error response from cache write-back.
0x13	External timeout. For example, timeout on interaction with another component.
0x14	Internal timeout. For example, timeout on interface within the component.
0x15	Deferred error from Completer not supported at Requester. For example, poisoned data received from the Completer of an access by a Requester that cannot defer the error further.
0x16	Deferred error from Requester not supported at Completer. For example, poisoned data received from the Requester of an access by a Completer that cannot defer the error further.
0x17	Deferred error from Completer passed through. For example, poisoned data received from the Completer of an access and returned to the Requester.
0x18	Deferred error from Requester passed through. For example, poisoned data received from the Requester of an access and deferred to the Completer.
0x19	Error recorded by PCIe error logs. Indicates that the component has recorded an error in a PCIe error log. This might be the PCIe device status register, AER, DVSEC, or other mechanisms defined by PCIe.
0x1A	Other internal error. For example, parity error on internal state of the component that is not covered by another primary error code.

All other values are reserved.

The implemented set of valid values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **UNKNOWN/WI** if all the following are true:
  - the node that owns error record n does not implement the Common Fault Injection Model Extension.
  - ERR<n>STATUS.V == 0.
- Access to this field is **UNKNOWN/WI** if all the following are true:
  - ERRPFGF[FirstRecordOfNode(n)].SYN == 0.
  - ERR<n>STATUS.V == 0.

- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.DE == 0.
  - ERR<n>STATUS.UE == 0.
  - ERR<n>STATUS.CE != 0b00.
  - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE == 0.
  - ERR<n>STATUS.DE != 0.
  - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write.
- Access to this field is **RO** if all the following are true:
  - ERR<n>STATUS.UE != 0.
  - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write.
- Otherwise, access to this field is **RW**.

Accessing ERR<n>STATUS

ERR<n>STATUS.{AV, V, UE, ER, OF, MV, CE, DE, PN, UET, CI} are write-one-to-clear (W1C) fields, meaning writes of zero are ignored, and a write of one or all-ones to the field clears the field to zero. ERR<n>STATUS.{IERR, SERR} are read/write (RW) fields, although the set of implemented valid values is IMPLEMENTATION DEFINED. See also ERR<n>PFGF.SYN.

After reading ERR<n>STATUS, software must clear the valid fields in the register to allow new errors to be recorded. However, between reading the register and clearing the valid fields, a new error might have overwritten the register. To prevent this error being lost by software, the register prevents updates to fields that might have been updated by a new error.

When RAS System Architecture v1.0 is implemented:

- Writes to ERR<n>STATUS.{UE, DE, CE} are ignored if ERR<n>STATUS.OF is 1 and is not being cleared to 0.
- Writes to ERR<n>STATUS.V are ignored if any of ERR<n>STATUS.{UE, DE, CE} are nonzero and are not being cleared to zero.
- Writes to ERR<n>STATUS.{AV, MV} and the ERR<n>STATUS.{ER, PN, UET, IERR, SERR} syndrome fields are ignored if the highest priority nonzero error status field is not being cleared to zero. The error status fields in priority order from highest to lowest, are ERR<n>STATUS.UE, ERR<n>STATUS.DE, and ERR<n>STATUS.CE.

When RAS System Architecture v1.1 is implemented, a write to the register is ignored if all of:

- Any of ERR<n>STATUS.{V, UE, OF, CE, DE} are nonzero before the write.
- The write does not clear the nonzero ERR<n>STATUS.{V, UE, OF, CE, DE} fields to zero by writing ones to the applicable field or fields.

Some of the fields in ERR<n>STATUS are also defined as UNKNOWN where certain combinations of ERR<n>STATUS.{V, DE, UE} are zero. The rules for writes to ERR<n>STATUS allow a node to implement such a field as a fixed read-only value.

For example, when RAS System Architecture v1.1 is implemented, a write to ERR<n>STATUS when ERR<n>STATUS.V is 1 results in either ERR<n>STATUS.V field being cleared to zero, or ERR<n>STATUS.V not changing. Since all fields in ERR<n>STATUS, other than ERR<n>STATUS.{AV, V, MV}, usually read as UNKNOWN values when ERR<n>STATUS.V is zero, this means those fields can be implemented as read-only if applicable.

To ensure correct and portable operation, when software is clearing the valid fields in the register to allow new errors to be recorded, Arm recommends that software performs the following sequence of operations in order:

1. Read ERR<n>STATUS and determine which fields need to be cleared to zero.
2. In a single write to ERR<n>STATUS:
   - Write ones to all the W1C fields that are nonzero in the read value.
   - Write zero to all the W1C fields that are zero in the read value.
   - Write zero to all the RW fields.
3. Read back ERR<n>STATUS after the write to confirm no new fault has been recorded.

Otherwise, these fields might not have the correct value when a new fault is recorded.

ERR<n>STATUS can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0x010` + (64 * n)	ERR<n>STATUS

Accessible as follows:

- When ERR<n>STATUS.V != 0, ERR<n>STATUS.V is not being cleared to 0b0 in the same write, and RAS System Architecture v1p1 is implemented, accesses to this register are **RO**.
- When ERR<n>STATUS.UE != 0, ERR<n>STATUS.UE is not being cleared to 0b0 in the same write, and RAS System Architecture v1p1 is implemented, accesses to this register are **RO**.

- When ERR<n>STATUS.OF != 0, ERR<n>STATUS.OF is not being cleared to 0b0 in the same write, and RAS System Architecture v1p1 is implemented, accesses to this register are **RO**.
- When ERR<n>STATUS.CE != 0b00, ERR<n>STATUS.CE is not being cleared to 0b00 in the same write, and RAS System Architecture v1p1 is implemented, accesses to this register are **RO**.
- When ERR<n>STATUS.DE != 0, ERR<n>STATUS.DE is not being cleared to 0b0 in the same write, and RAS System Architecture v1p1 is implemented, accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRPIDR0, Peripheral Identification Register 0

The ERRPIDR0 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

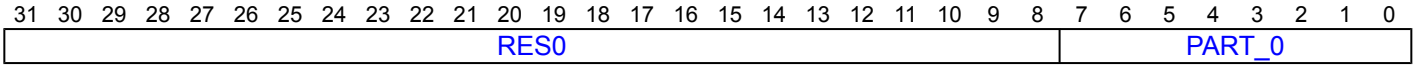
Implementation of this register is OPTIONAL.

ERRPIDR0 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRPIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, bits [7:0].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, then it is stored in ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 8 bits, ERRPIDR2.REVISION and ERRPIDR3.REVAND, available to define the revision of the component.
- If a 16-bit part number is used, then it is stored in ERRPIDR2.PART_2, ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 4 bits, ERRPIDR3.REVISION, available to define the revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRPIDR0

This section shows the offset of ERRPIDR0 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRPIDR0.

ERRPIDR0 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xFE0	ERRPIDR0

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRPIDR1, Peripheral Identification Register 1

The ERRPIDR1 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

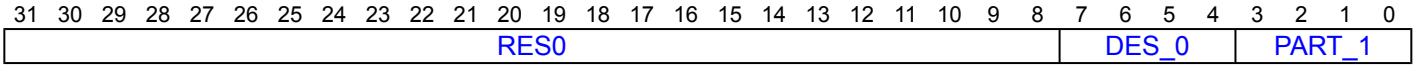
Implementation of this register is OPTIONAL.

ERRPIDR1 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRPIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, JEP106 identification code, bits [3:0]. ERRPIDR1.DES_0 and ERRPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

Note

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, bits [11:8].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, then it is stored in ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 8 bits, ERRPIDR2.REVISION and ERRPIDR3.REVAND, available to define the revision of the component.
- If a 16-bit part number is used, then it is stored in ERRPIDR2.PART_2, ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 4 bits, ERRPIDR3.REVISION, available to define the revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRPIDR1

This section shows the offset of ERRPIDR1 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRPIDR1.

ERRPIDR1 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xFE4	ERRPIDR1

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRPIDR2, Peripheral Identification Register 2

The ERRPIDR2 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

Implementation of this register is OPTIONAL.

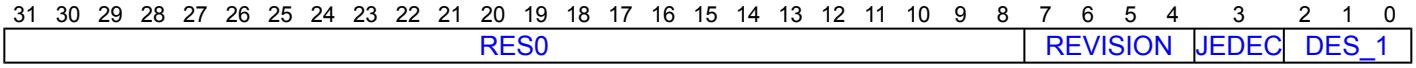
ERRPIDR2 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRPIDR2 is a 32-bit register.

Field descriptions

When the component uses a 12-bit part number:



Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Component major revision. ERRPIDR2.REVISION and ERRPIDR3.REVAND together form the revision number of the component, with ERRPIDR2.REVISION being the most significant part and ERRPIDR3.REVAND the least significant part. When a component is changed, ERRPIDR2.REVISION or ERRPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. ERRPIDR3.REVAND should be set to `0b0000` when ERRPIDR2.REVISION is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as `0b1`.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, JEP106 identification code, bits [6:4]. ERRPIDR1.DES_0 and ERRPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

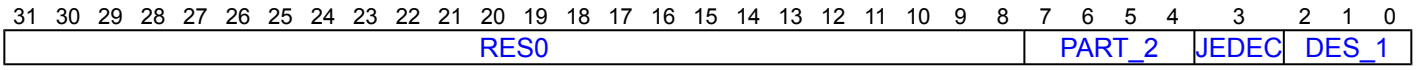
Note

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

When the component uses a 16-bit part number:



Bits [31:8]

Reserved, RES0.

PART_2, bits [7:4]

Part number, bits [15:12].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, then it is stored in ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 8 bits, ERRPIDR2.REVISION and ERRPIDR3.REVAND, available to define the revision of the component.
- If a 16-bit part number is used, then it is stored in ERRPIDR2.PART_2, ERRPIDR1.PART_1 and ERRPIDR0.PART_0. There are 4 bits, ERRPIDR3.REVISION, available to define the revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as `0b1`.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, JEP106 identification code, bits [6:4]. ERRPIDR1.DES_0 and ERRPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

Note

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRPIDR2

This section shows the offset of ERRPIDR2 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRPIDR2.

ERRPIDR2 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	0xFE8	ERRPIDR2

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRPIDR3, Peripheral Identification Register 3

The ERRPIDR3 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

Implementation of this register is OPTIONAL.

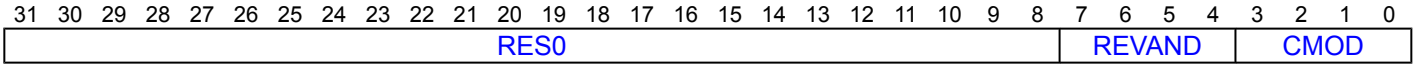
ERRPIDR3 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRPIDR3 is a 32-bit register.

Field descriptions

When the component uses a 12-bit part number:



Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Component minor revision. ERRPIDR2.REVISION and ERRPIDR3.REVAND together form the revision number of the component, with ERRPIDR2.REVISION being the most significant part and ERRPIDR3.REVAND the least significant part. When a component is changed, ERRPIDR2.REVISION or ERRPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. ERRPIDR3.REVAND should be set to `0b0000` when ERRPIDR2.REVISION is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer Modified.

Indicates the component has been modified.

A value of `0b0000` means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

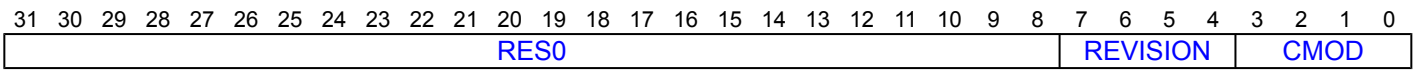
For any two components with the same Unique Component Identifier:

- If ERRPIDR3.CMOD is zero in both components, then the components are identical.
- If ERRPIDR3.CMOD has the same nonzero value in both components, then this does not necessarily mean that they have the same modifications.
- If ERRPIDR3.CMOD is nonzero in either component, the two components might not be identical despite having the same Unique Component Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

When the component uses a 16-bit part number:



Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Component revision. When a component is changed, ERRPIDR3.REVISION is increased to ensure that software can differentiate the different revisions of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer Modified.

Indicates the component has been modified.

A value of `0b0000` means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If ERRPIDR3.CMOD is zero in both components, then the components are identical.
- If ERRPIDR3.CMOD has the same nonzero value in both components, then this does not necessarily mean that they have the same modifications.
- If ERRPIDR3.CMOD is nonzero in either component, the two components might not be identical despite having the same Unique Component Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRPIDR3

This section shows the offset of ERRPIDR3 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRPIDR3.

ERRPIDR3 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0xFEC`	ERRPIDR3

Accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRPIDR4, Peripheral Identification Register 4

The ERRPIDR4 characteristics are:

Purpose

Provides discovery information about the component.

Configuration

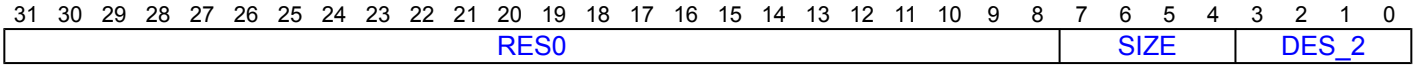
Implementation of this register is OPTIONAL.

ERRPIDR4 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRPIDR4 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

When RAS System Architecture v2 is implemented:

Size of the component.

SIZE	Meaning
0b0000	FEAT_RASSA_4KB is implemented.
0b0010	FEAT_RASSA_16KB is implemented.
0b0100	FEAT_RASSA_64KB is implemented.

All other values are reserved.

Otherwise:

Size of the component.

SIZE	Meaning
0b0000	One of the following is true: • The component uses a single 4KB block. • The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.
0b0001..0b1111	The component occupies $2^{ERRPIDR4.SIZE}$ 4KB blocks.

DES_2, bits [3:0]

Designer, JEP106 continuation code. This is the JEDEC-assigned JEP106 bank identifier for the designer of the component, minus 1. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

Note

For a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value `0x4`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRPIDR4

This section shows the offset of ERRPIDR4 when FEAT_RASSA_4KB_GRP is implemented. If FEAT_RASSA_16KB_GRP or FEAT_RASSA_64KB_GRP is implemented, see 'RAS memory-mapped register views' for the offset of ERRPIDR4.

ERRPIDR4 can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
RAS	`0xFD0`	ERRPIDR4

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_ABPR, CPU Interface Aliased Binary Point Register

The GICC_ABPR characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_ABPR are RES0.

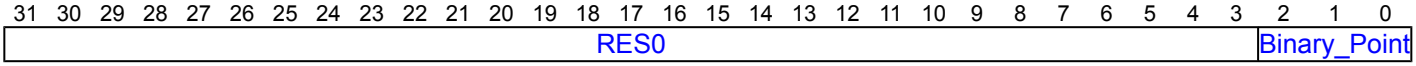
In systems that support two Security states:

- This register is an alias of the Non-secure copy of GICC_BPR.
- Non-secure accesses to this register return a shifted value of the binary point.
- If ICC_CTLR_EL3.CBPR_EL1NS == 1, Secure accesses to this register access ICC_BPR0_EL1.

Attributes

GICC_ABPR is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

Binary_Point, bits [2:0]

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The following list describes how this field determines the interrupt priority bits assigned to the group priority field:

- 'Secure ICC_BPR1_EL1 Binary Point when CBPR == 0' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069), for the processing of Group 1 interrupts in a GIC implementation that supports interrupt grouping, when GICC_CTLR.CBPR == 0.
- 'Non-secure ICC_BPR1_EL1 Binary Point when CBPR == 0' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069), for all other cases.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICC_ABPR

This register is used only when System register access is not enabled. When System register access is enabled, the System registers ICC_BPR0_EL1 and ICC_BPR1_EL1 provide equivalent functionality.

GICC_ABPR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x001C	GICC_ABPR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_AEOIR, CPU Interface Aliased End Of Interrupt Register

The GICC_AEOIR characteristics are:

Purpose

A write to this register performs priority drop for the specified Group 1 interrupt and, if the appropriate GICC_CTLR.EOImodeS or GICC_CTLR.EOImodeNS field == 0, also deactivates the interrupt.

Configuration

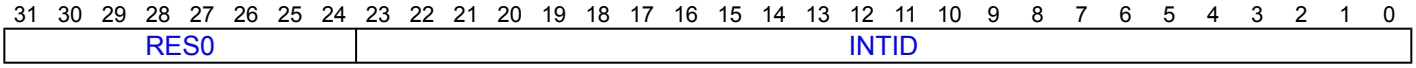
This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_AEOIR are RES0.

When GICD_CTLR.DS==0, this register is an alias of the Non-secure view of GICC_EOIR. A Secure access to this register is identical to a Non-secure access to GICC_EOIR.

Attributes

GICC_AEOIR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Accessing GICC_AEOIR

A write to this register must correspond to the most recently acknowledged Group 1 interrupt. If a value other than the last value read from GICC_AIAR is written to this register, the effect is UNPREDICTABLE.

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_EOIR1 provides equivalent functionality.
- For AArch64 implementations, ICC_EOIR1_EL1 provides equivalent functionality.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_AEOIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x0024	GICC_AEOIR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **WO**.
- When an access is Secure, accesses to this register are **WO**.
- When an access is Non-secure, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_AHPPIR, CPU Interface Aliased Highest Priority Pending Interrupt Register

The GICC_AHPPIR characteristics are:

Purpose

If the highest priority pending interrupt is in Group 1, this register provides the INTID of the highest priority pending interrupt on the CPU interface.

Configuration

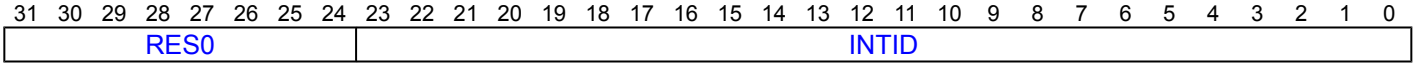
This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_AHPPIR are RES0.

If GICD_CTLR.DS==0, this register is an alias of the Non-secure view of GICC_HPPIR. A Secure access to this register is identical to a Non-secure access to GICC_HPPIR.

Attributes

GICC_AHPPIR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Accessing GICC_AHPPIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_HPPIR1 provides equivalent functionality.
- For AArch64 implementations, ICC_HPPIR1_EL1 provides equivalent functionality.

If the highest priority pending interrupt is in Group 0, a read of this register returns the special INTID 1023.

Interrupt identifiers corresponding to an interrupt group that is not enabled are ignored.

If the highest priority pending interrupt is a direct interrupt that is both individually enabled in the Distributor and part of an interrupt group that is enabled in the Distributor, and the interrupt group is disabled in the CPU interface for this PE, this register returns the special INTID 1023.

For more information about pending interrupts that are not considered when determining the highest priority pending interrupt, see 'Preemption' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_AHPPIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x0028	GICC_AHPPIR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_AIAR, CPU Interface Aliased Interrupt Acknowledge Register

The GICC_AIAR characteristics are:

Purpose

Provides the INTID of the signaled Group 1 interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

Configuration

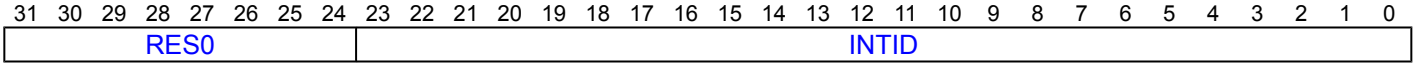
This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_AIAR are RES0.

When GICD_CTLR.DS==0, this register is an alias of the Non-secure view of GICC_IAR. A Secure access to this register is identical to a Non-secure access to GICC_IAR.

Attributes

GICC_AIAR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Accessing GICC_AIAR

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_AIAR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x0020	GICC_AIAR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_APR<n>, CPU Interface Active Priorities Registers, n = 0 - 3

The GICC_APR<n> characteristics are:

Purpose

Provides information about interrupt active priorities.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_APR<n> are RES0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value `0x00000000` is consistent with no interrupts being active.

When GICD_CTLR.DS == 0, these registers are Banked, and Non-secure accesses do not affect Secure operation. The Secure copies of these registers hold active priorities for Group 0 interrupts, and the Non-secure copies provide a Non-secure view of the active priorities for Group 1 interrupts.

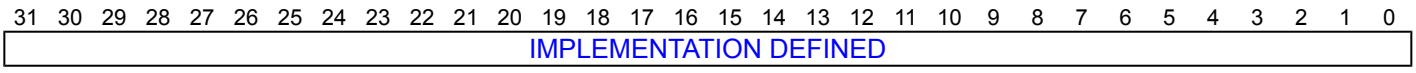
GICC_APR1 is implemented only in implementations that support 6 or more bits of priority. GICC_APR2 and GICC_APR3 are implemented only in implementations that support 7 bits of priority.

When GICD_CTLR.DS==1, these registers hold the active priorities for Group 0 interrupts, and the active priorities for Group 1 interrupts are held by the GICC_NSAPR<n> registers.

Attributes

GICC_APR<n> is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'00000000000000000000000000000000'`.

Accessing GICC_APR<n>

These registers are used only when System register access is not enabled. When System register access is enabled the following registers provide equivalent functionality:

- In AArch64:
  - For Group 0, ICC_AP0R<n>_EL1.
  - For Group 1, ICC_AP1R<n>_EL1.
- In AArch32:
  - For Group 0, ICC_AP0R<n>.
  - For Group 1, ICC_AP1R<n>.

GICC_APR<n> can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	`0x00D0` + (4 * n)	GICC_APR<n>

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_BPR, CPU Interface Binary Point Register

The GICC_BPR characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_BPR are RES0.

In systems that support two Security states:

- This register is Banked.
- The Secure instance of this register determines Group 0 interrupt preemption.
- The Non-secure instance of this register determines Group 1 interrupt preemption.

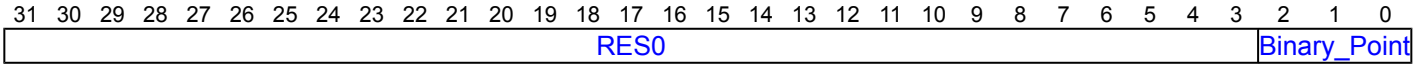
In systems that support only one Security state, when GICC_CTLR.CBPR == 0, this register determines only Group 0 interrupt preemption.

When GICC_CTLR.CBPR == 1, this register determines interrupt preemption for both Group 0 and Group 1 interrupts.

Attributes

GICC_BPR is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

Binary_Point, bits [2:0]

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The following list describes how this field determines the interrupt priority bits assigned to the group priority field:

- 'Secure ICC_BPR1_EL1 Binary Point when CBPR == 0' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069), for the processing of Group 1 interrupts in a GIC implementation that supports interrupt grouping, when GICC_CTLR.CBPR == 0.
- 'Non-secure ICC_BPR1_EL1 Binary Point when CBPR == 0' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069), for all other cases.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information

Note

Aliasing the Non-secure GICC_BPR as GICC_ABPR in a multiprocessor system permits a PE that can make only Secure accesses to configure the preemption setting for Group 1 interrupts by accessing GICC_ABPR.

Accessing GICC_BPR

This register is used only when System register access is not enabled. When System register access is enabled this register is RAZ/WI, and the System registers ICC_BPR0_EL1 and ICC_BPR1_EL1 provide equivalent functionality.

GICC_BPR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x0008	GICC_BPR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_CTLR, CPU Interface Control Register

The GICC_CTLR characteristics are:

Purpose

Controls the CPU interface, including enabling of interrupt groups, interrupt signal bypass, binary point registers used, and separation of priority drop and interrupt deactivation.

Note

If the GIC implementation supports two Security states, independent EOI controls are provided for accesses from each Security state. Secure accesses handle both Group 0 and Group 1 interrupts, and Non-secure accesses handle Group 1 interrupts only.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_CTLR are RES0.

In a GIC implementation that supports two Security states:

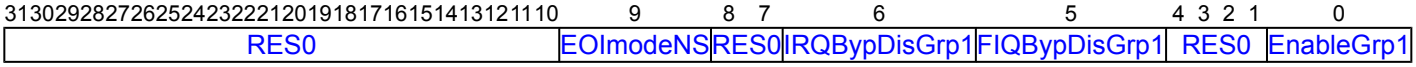
- This register is Banked.
- The register bit assignments are different in the Secure and Non-secure copies.

Attributes

GICC_CTLR is a 32-bit register.

Field descriptions

When GICD_CTLR.DS==0, Non-secure access:



Bits [31:10]

Reserved, RES0.

EOImodeNS, bit [9]

Controls the behavior of Non-secure accesses to GICC_EOIR, GICC_AEOIR, and GICC_DIR.

EOImodeNS	Meaning
0b0	GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE.
0b1	GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality.

Note

An implementation is permitted to make this bit RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [8:7]

Reserved, RES0.

IRQBypDisGrp1, bit [6]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 1:

IRQBypDisGrp1	Meaning
0b0	The bypass IRQ signal is signaled to the PE.
0b1	The bypass IRQ signal is not signaled to the PE.

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

FIQBypDisGrp1, bit [5]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 1:

FIQBypDisGrp1	Meaning
0b0	The bypass FIQ signal is signaled to the PE.
0b1	The bypass FIQ signal is not signaled to the PE.

If System register access is enabled for EL3 and ICC_SRE_EL3.DFB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [4:1]

Reserved, RES0.

EnableGrp1, bit [0]

This Non-secure field enables the signaling of Group 1 interrupts by the CPU interface to a target PE:

EnableGrp1	Meaning
0b0	Group 1 interrupt signaling is disabled.
0b1	Group 1 interrupt signaling is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When GICD_CTLR.DS==0, Secure access:

31302928272625242322212019181716151413121	10	9	8	7	6	5
RES0	EOImodeNS	EOImodeS	IRQBypDisGrp1	FIQBypDisGrp1	IRQBypDisGrp0	FIQBypDisGr

Bits [31:11]

Reserved, RES0.

EOImodeNS, bit [10]

Controls the behavior of Non-secure accesses to GICC_EOIR, GICC_AEOIR, and GICC_DIR.

EOImodeNS	Meaning
0b0	GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE.
0b1	GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality.

Note

An implementation is permitted to make this bit RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

EOImodeS, bit [9]

Controls the behavior of Secure accesses to GICC_EOIR, GICC_AEOIR, and GICC_DIR.

EOImodeS	Meaning
0b0	GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE.
0b1	GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality.

Note

An implementation is permitted to make this bit RAO/WI.

This field shares state with GICC_CTLR.EOImode.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

IRQBypDisGrp1, bit [8]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 1:

IRQBypDisGrp1	Meaning
0b0	The bypass IRQ signal is signaled to the PE.
0b1	The bypass IRQ signal is not signaled to the PE.

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

FIQBypDisGrp1, bit [7]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 1:

FIQBypDisGrp1	Meaning
`0b0`	The bypass FIQ signal is signaled to the PE.
`0b1`	The bypass FIQ signal is not signaled to the PE.

If System register access is enabled for EL3 and ICC_SRE_EL3.DFB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

IRQBypDisGrp0, bit [6]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 0:

IRQBypDisGrp0	Meaning
`0b0`	The bypass IRQ signal is signaled to the PE.
`0b1`	The bypass IRQ signal is not signaled to the PE.

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

FIQBypDisGrp0, bit [5]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 0:

FIQBypDisGrp0	Meaning
`0b0`	The bypass FIQ signal is signaled to the PE.
`0b1`	The bypass FIQ signal is not signaled to the PE.

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

CBPR, bit [4]

Controls whether GICC_BPR provides common control of preemption to Group 0 and Group 1 interrupts:

CBPR	Meaning
`0b0`	GICC_BPR determines preemption for Group 0 interrupts only. GICC_ABPR determines preemption for Group 1 interrupts.
`0b1`	GICC_BPR determines preemption for both Group 0 and Group 1 interrupts.

This field is an alias of ICC_CTLR_EL3.CBPR_EL1NS.

In a GIC that supports two Security states, when CBPR == 1:

- A Non-secure read of GICC_BPR returns the value of Secure GICC_BPR.Binary_Point, incremented by 1, and saturated to `0b111`.
- Non-secure writes of GICC_BPR are ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

FIQEn, bit [3]

Controls whether the CPU interface signals Group 0 interrupts to a target PE using the FIQ or IRQ signal:

FIQEn	Meaning
`0b0`	Group 0 interrupts are signaled using the IRQ signal.
`0b1`	Group 0 interrupts are signaled using the FIQ signal.

Group 1 interrupts are signaled using the IRQ signal only.

If an implementation supports two Security states, this bit is permitted to be RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Bit [2]

Reserved, RES0.

EnableGrp1, bit [1]

This Non-secure field enables the signaling of Group 1 interrupts by the CPU interface to a target PE:

EnableGrp1	Meaning
`0b0`	Group 1 interrupt signaling is disabled.
`0b1`	Group 1 interrupt signaling is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

EnableGrp0, bit [0]

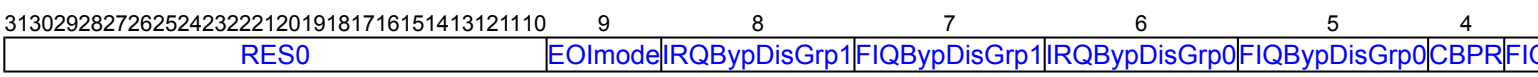
Enables the signaling of Group 0 interrupts by the CPU interface to a target PE:

EnableGrp0	Meaning
0b0	Group 0 interrupt signaling is disabled.
0b1	Group 0 interrupt signaling is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When GICD_CTLR.DS == 1:



Bits [31:10]

Reserved, RES0.

EOImode, bit [9]

Controls the behavior of accesses to GICC_EOIR, GICC_AEOIR, and GICC_DIR.

EOImode	Meaning
0b0	GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE.
0b1	GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality.

Note

An implementation is permitted to make this bit RAO/WI.

This field shares state with GICC_CTLR.EOImodeS.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

IRQBypDisGrp1, bit [8]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 1:

IRQBypDisGrp1	Meaning
0b0	The bypass IRQ signal is signaled to the PE.
0b1	The bypass IRQ signal is not signaled to the PE.

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

FIQBypDisGrp1, bit [7]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 1:

FIQBypDisGrp1	Meaning
0b0	The bypass FIQ signal is signaled to the PE.
0b1	The bypass FIQ signal is not signaled to the PE.

If System register access is enabled for EL3 and ICC_SRE_EL3.DFB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

IRQBypDisGrp0, bit [6]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 0:

IRQBypDisGrp0	Meaning
0b0	The bypass IRQ signal is signaled to the PE.
0b1	The bypass IRQ signal is not signaled to the PE.

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

FIQBypDisGrp0, bit [5]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 0:

FIQBypDisGrp0	Meaning
0b0	The bypass FIQ signal is signaled to the PE.
0b1	The bypass FIQ signal is not signaled to the PE.

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

CBPR, bit [4]

Controls whether GICC_BPR provides common control of preemption to Group 0 and Group 1 interrupts:

CBPR	Meaning
0b0	GICC_BPR determines preemption for Group 0 interrupts only. GICC_ABPR determines preemption for Group 1 interrupts.
0b1	GICC_BPR determines preemption for both Group 0 and Group 1 interrupts.

This field is an alias of ICC_CTLR_EL3.CBPR_EL1NS.

In a GIC that supports two Security states, when CBPR == 1:

- A Non-secure read of GICC_BPR returns the value of Secure GICC_BPR.Binary_Point, incremented by 1, and saturated to 0b111.
- Non-secure writes of GICC_BPR are ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

FIQEn, bit [3]

Controls whether the CPU interface signals Group 0 interrupts to a target PE using the FIQ or IRQ signal:

FIQEn	Meaning
0b0	Group 0 interrupts are signaled using the IRQ signal.
0b1	Group 0 interrupts are signaled using the FIQ signal.

Group 1 interrupts are signaled using the IRQ signal only.

If an implementation supports two Security states, this bit is permitted to be RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [2]

Reserved, RES0.

EnableGrp1, bit [1]

This Non-secure field enables the signaling of Group 1 interrupts by the CPU interface to a target PE:

EnableGrp1	Meaning
0b0	Group 1 interrupt signaling is disabled.
0b1	Group 1 interrupt signaling is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

EnableGrp0, bit [0]

Enables the signaling of Group 0 interrupts by the CPU interface to a target PE:

EnableGrp0	Meaning
0b0	Group 0 interrupt signaling is disabled.
0b1	Group 0 interrupt signaling is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing GICC_CTLR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_CTLR and ICC_MCTLR provide equivalent functionality.
- For AArch64 implementations, ICC_CTLR_EL1 and ICC_CTLR_EL3 provide equivalent functionality.

GICC_CTLR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x0000	GICC_CTLR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_DIR, CPU Interface Deactivate Interrupt Register

The GICC_DIR characteristics are:

Purpose

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified interrupt.

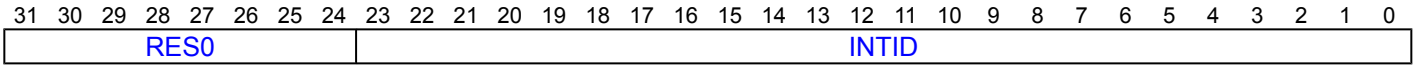
Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_DIR are RES0.

Attributes

GICC_DIR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Accessing GICC_DIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_DIR provides equivalent functionality.
- For AArch64 implementations, ICC_DIR_EL1 provides equivalent functionality.

Writes to this register have an effect only in the following cases:

- When GICD_CTLR.DS == 1, if GICC_CTLR.EOImode == 1.
- In GIC implementations that support two Security states:
  - If the access is Secure and GICC_CTLR.EOImodeS == 1.
  - If the access is Non-secure and GICC_CTLR.EOImodeNS == 1.

The following writes must be ignored:

- Writes to this register when the corresponding EOImode field in GICC_CTLR == 0. In systems that support system error generation, an implementation might generate a system error.

- Writes to this register when the corresponding EOImode field in GICC_CTLR == 0 and the corresponding interrupt is not active. In systems that support system error generation, an implementation might generate a system error. In implementations using the GIC Stream Protocol Interface, these writes correspond to a Deactivate packet for an interrupt that is not active. For more information, see 'Deactivate (ICC)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

If the corresponding EOImode field in GICC_CTLR is 1 and this register is written to without a corresponding write to GICC_EOIR or GICC_AEOIR, the interrupt is deactivated but the bit corresponding to it in the active priorities registers remains set.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_DIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x1000	GICC_DIR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **WO**.
- When an access is Secure, accesses to this register are **WO**.
- When an access is Non-secure, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_EOIR, CPU Interface End Of Interrupt Register

The GICC_EOIR characteristics are:

Purpose

A write to this register performs priority drop for the specified interrupt and, if the appropriate GICC_CTLR.EOImodeS or GICC_CTLR.EOImodeNS field == 0, also deactivates the interrupt.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_EOIR are RES0.

If GICD_CTLR.DS==0:

- This register is Common.
- GICC_AEOIR is an alias of the Non-secure view of this register.

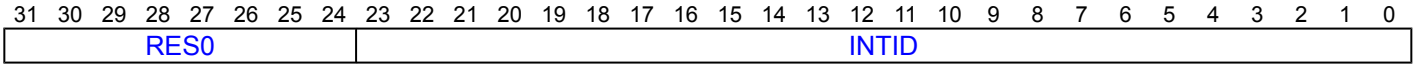
For Secure writes when GICD_CTLR.DS==0, or for Secure and Non-secure writes when GICD_CTLR.DS==1, the register provides functionality for Group 0 interrupts.

For Non-secure writes when GICD_CTLR.DS==1, the register provides functionality for Group 1 interrupts.

Attributes

GICC_EOIR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Additional information

For every read of a valid INTID from GICC_IAR, the connected PE must perform a matching write to GICC_EOIR. The value written to GICC_EOIR must be the INTID from GICC_IAR. Reads of INTIDs 1020-1023 do not require matching writes.

Note

Arm recommends that software preserves the entire register value read from GICC_IAR, and writes that value back to GICC_EOIR on completion of interrupt processing.

For nested interrupts, the order of writes to this register must be the reverse of the order of interrupt acknowledgment. Behavior is UNPREDICTABLE if:

- This ordering constraint is not maintained.
- The value written to this register does not match an active interrupt, or the ID of a spurious interrupt.
- The value written to this register does not match the last valid interrupt value read from GICC_IAR.

For general information about the effect of writes to end of interrupt registers, and about the possible separation of the priority drop and interrupt deactivate operations, see 'Interrupt lifecycle' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

If GICD_CTLR.DS==0:

- GICC_CTLR.EOImodeS controls the behavior of Secure accesses to GICC_EOIR and GICC_AEOIR.
- GICC_CTLR.EOImodeNS controls the behavior of Non-secure accesses to GICC_EOIR and GICC_AEOIR.

Accessing GICC_EOIR

The following writes must be ignored:

- Writes of INTIDs 1020-1023.
- Secure writes corresponding to Group 1 interrupts. In systems that support system error generation, an implementation might generate a system error. In this case, GIC behavior is predictable, and the highest Secure active priority (in the Secure copy of GICC_APR<n>) will be reset if the highest active priority is Secure. System behavior is UNPREDICTABLE.
- Non-secure writes corresponding to Group 0 interrupts when GICC_CTLR.EOImodeS == 1. In systems that support system error generation, an implementation might generate a system error. In this case, GIC behavior is predictable, and the highest Non-secure active priority (in the Non-secure copy of GICC_APR<n>) will be reset if the highest active priority is Non-secure. System behavior is UNPREDICTABLE.

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_EOIR0 and ICC_EOIR1 provide equivalent functionality.
- For AArch64 implementations, ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide equivalent functionality.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_EOIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x0010	GICC_EOIR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **WO**.
- When an access is Secure, accesses to this register are **WO**.
- When an access is Non-secure, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_HPPIR, CPU Interface Highest Priority Pending Interrupt Register

The GICC_HPPIR characteristics are:

Purpose

Provides the INTID of the highest priority pending interrupt on the CPU interface.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_HPPIR are RES0.

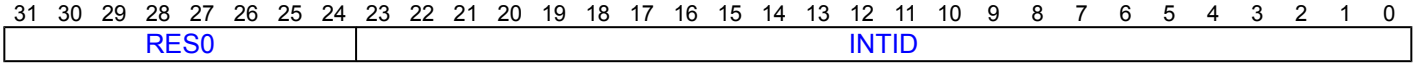
If GICD_CTLR.DS==0:

- This register is Common.
- GICC_AHPPIR is an alias of the Non-secure view of this register.

Attributes

GICC_HPPIR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Accessing GICC_HPPIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_HPPIR0 and ICC_HPPIR1 provide equivalent functionality.
- For AArch64 implementations, ICC_HPPIR0_EL1 and ICC_HPPIR1_EL1 provide equivalent functionality.

If the highest priority pending interrupt is in Group 0, a Non-secure read of this register returns the special INTID 1023.

For Secure reads when GICD_CTLR.DS==0, or for Secure and Non-secure reads when GICD_CTLR.DS==1, returns the special INTID 1022 if the highest priority pending interrupt is in Group 1.

If no interrupts are in the pending state, a read of this register returns the special INTID 1023.

Interrupt identifiers corresponding to an interrupt group that is not enabled are ignored.

If the highest priority pending interrupt is a direct interrupt that is both individually enabled in the Distributor and part of an interrupt group that is enabled in the Distributor, and the interrupt group is disabled in the CPU interface for this PE, this register returns the special INTID 1023.

For more information about pending interrupts that are not considered when determining the highest priority pending interrupt, see 'Preemption' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_HPPIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x0018	GICC_HPPIR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_IAR, CPU Interface Interrupt Acknowledge Register

The GICC_IAR characteristics are:

Purpose

Provides the INTID of the signaled interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_IAR are RES0.

This register is available in all configurations of the GIC. If GICD_CTLR.DS==0:

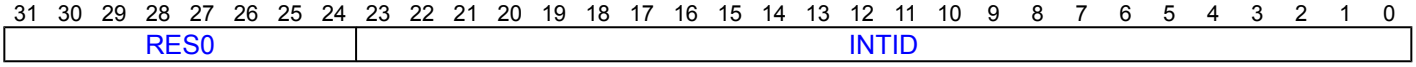
- This register is Common.
- GICC_AIAR is an alias of the Non-secure view of this register.

The format of the INTID is governed by whether affinity routing is enabled for a Security state.

Attributes

GICC_IAR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Additional information

A read of this register returns the INTID of the highest priority pending interrupt for the CPU interface. The read returns a spurious INTID of 1023 if any of the following apply:

- Forwarding of interrupts by the Distributor to the CPU interface is disabled.
- Signaling of interrupts by the CPU interface to the connected PE is disabled.
- There are no pending interrupts on the CPU interface with sufficient priority for the interface to signal it to the PE.

When the GIC returns a valid INTID to a read of this register it treats the read as an acknowledge of that interrupt. In addition, it changes the interrupt status from pending to active, or to active and pending if the pending state of the interrupt persists. Normally, the pending state of an interrupt persists only if the interrupt is level-sensitive and remains asserted.

For every read of a valid INTID from GICC_IAR, the connected PE must perform a matching write to GICC_EOIR.

Note

- Arm recommends that software preserves the entire register value read from this register, and writes that value back to GICC_EOIR on completion of interrupt processing.
- For SPIs, although multiple target PEs might attempt to read this register at any time, only one PE can obtain a valid INTID. For more information, see 'Activation' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accessing GICC_IAR

When GICD_CTLR.DS==1, if the highest priority pending interrupt is in Group 1, the special INTID 1022 is returned.

In GIC implementations that support two Security states, if the highest priority pending interrupt is in Group 0, Non-secure reads return the special INTID 1023.

In GIC implementations that support two Security states, if the highest priority pending interrupt is in Group 1, Secure reads return the special INTID 1022.

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_IAR0 and ICC_IAR1 provide equivalent functionality.
- For AArch64 implementations, ICC_IAR0_EL1 and ICC_IAR1_EL1 provide equivalent functionality.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_IAR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x000C	GICC_IAR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_IIDR, CPU Interface Identification Register

The GICC_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the CPU interface.

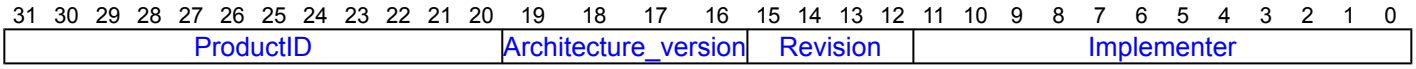
Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_IIDR are RES0.

Attributes

GICC_IIDR is a 32-bit register.

Field descriptions



ProductID, bits [31:20]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Architecture_version, bits [19:16]

The version of the GIC architecture that is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Architecture_version	Meaning
0b0001	GICv1.
0b0010	GICv2.
0b0011	FEAT_GICv3 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1.
0b0100	FEAT_GICv4 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1.

Other values are reserved.

Access to this field is **RO**.

Revision, bits [15:12]

Revision number for the CPU interface.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the CPU interface.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for GICC_IIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GICC_IIDR

GICC_IIDR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	`0x00FC`	GICC_IIDR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_NSAPR<n>, CPU Interface Non-secure Active Priorities Registers, n = 0 - 3

The GICC_NSAPR<n> characteristics are:

Purpose

Provides information about Group 1 interrupt active priorities.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_NSAPR<n> are RES0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value `0x00000000` is consistent with no interrupts being active.

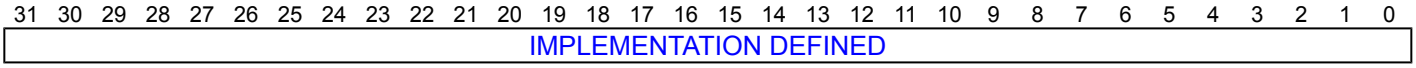
When GICD_CTLR.DS==0, these registers are RAZ/WI to Non-secure accesses.

GICC_NSAPR1 is implemented only in implementations that support 6 or more bits of priority. GICC_NSAPR2 and GICC_NSAPR3 are implemented only in implementations that support 7 bits of priority.

Attributes

GICC_NSAPR<n> is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'00000000000000000000000000000000'`.

Accessing GICC_NSAPR<n>

GICC_NSAPR<n> can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	`0x00E0` + (4 * n)	GICC_NSAPR<n>

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_PMR, CPU Interface Priority Mask Register

The GICC_PMR characteristics are:

Purpose

This register provides an interrupt priority filter. Only interrupts with a higher priority than the value in this register are signaled to the PE.

Note

Higher interrupt priority corresponds to a lower value of the Priority field.

Configuration

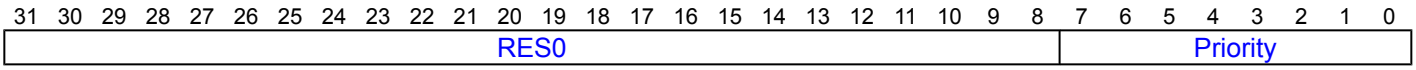
This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_PMR are RES0.

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states this register is Common.

Attributes

GICC_PMR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the CPU interface. If the priority of the interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

If the GIC implementation supports fewer than 256 priority levels some bits might be RAZ/WI, as follows:

- For 128 supported levels, bit [0] = `0b0`.
- For 64 supported levels, bits [1:0] = `0b00`.
- For 32 supported levels, bits [2:0] = `0b000`.
- For 16 supported levels, bits [3:0] = `0b0000`.

For more information, see 'Interrupt prioritization' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICC_PMR

If the GIC implementation supports two Security states:

- Non-secure accesses to this register can only read or write values corresponding to the lower half of the priority range.
- If a Secure write has programmed the register with a value that corresponds to a value in the upper half of the priority range then:

- Any Non-secure read of the register returns `0x00`, regardless of the value held in the register.
- Non-secure writes are ignored.

For more information, see 'Interrupt prioritization' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

GICC_PMR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	`0x0004`	GICC_PMR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_RPR, CPU Interface Running Priority Register

The GICC_RPR characteristics are:

Purpose

This register indicates the running priority of the CPU interface.

Configuration

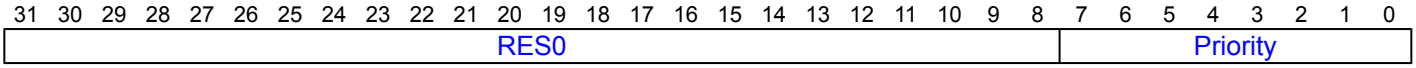
This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_RPR are RES0.

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states this register is Common.

Attributes

GICC_RPR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the CPU interface. This is the group priority of the current active interrupt.

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR was set to the minimum value.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GICC_RPR

If there is no active interrupt on the CPU interface, the idle priority value is returned.

If the GIC implementation supports two Security states, a Non-secure read of the Priority field returns:

- `0x00` if the field value is less than `0x80`.
- The Non-secure view of the Priority value if the field value is `0x80` or more.

For more information, see 'Interrupt prioritization' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Software cannot determine the number of implemented priority bits from this register.

GICC_RPR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x0014	GICC_RPR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_STATUSR, CPU Interface Status Register

The GICC_STATUSR characteristics are:

Purpose

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_STATUSR are RES0.

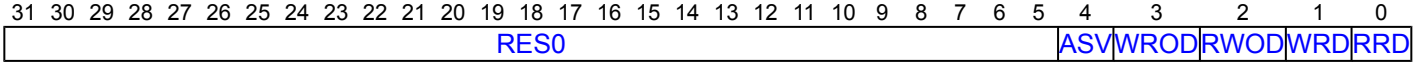
If the GIC implementation supports two Security states this register is Banked to provide Secure and Non-secure copies.

This register is used only when System register access is not enabled. If System register access is enabled, this register is not updated. Equivalent functionality might be provided by appropriate traps and exceptions.

Attributes

GICC_STATUSR is a 32-bit register.

Field descriptions



Bits [31:5]

Reserved, RES0.

ASV, bit [4]

Attempted security violation.

ASV	Meaning
0b0	Normal operation.
0b1	A Non-secure access to a Secure register has been detected.

Note

This bit is not set to 1 for registers where any of the fields are Non-secure.

WROD, bit [3]

Write to an RO location.

WROD	Meaning
0b0	Normal operation.
0b1	A write to an RO location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

RWOD, bit [2]

Read of a WO location.

RWOD	Meaning
0b0	Normal operation.
0b1	A read of a WO location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

WRD, bit [1]

Write to a reserved location.

WRD	Meaning
0b0	Normal operation.
0b1	A write to a reserved location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

RRD	Meaning
0b0	Normal operation.
0b1	A read of a reserved location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

Accessing GICC_STATUSR

This is an optional register. If the register is not implemented, the location is RAZ/WI.

If this register is implemented, GICV_STATUSR must also be implemented.

GICC_STATUSR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC CPU interface	0x002C	GICC_STATUSR (S)

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.

Component	Offset	Instance
GIC CPU interface	0x002C	GICC_STATUSR (NS)

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_CLRSPI_NSR, Clear Non-secure SPI Pending Register

The GICD_CLRSPI_NSR characteristics are:

Purpose

Removes the pending state from a valid SPI if permitted by the Security state of the access and the GICD_NSACR<n> value for that SPI.

A write to this register changes the state of a pending SPI to inactive, and the state of an active and pending SPI to active.

Configuration

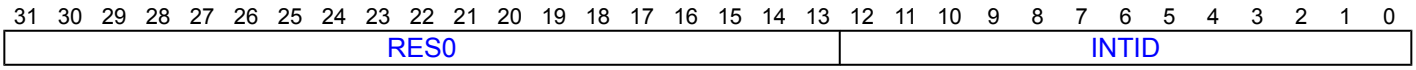
If GICD_TYPER.MBIS == 0, this register is reserved.

When GICD_CTLR.DS == 1, this register provides functionality for all SPIs.

Attributes

GICD_CLRSPI_NSR is a 32-bit register.

Field descriptions



Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

The INTID of the SPI.

Additional information

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to GICD_SETSPI_NSR or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to GICD_CLRSPI_NSR, GICD_CLRSPI_SR, or GICD_ICPENDR<n>.
- For a level-sensitive interrupt, a write to GICD_SETSPI_NSR or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to GICD_CLRSPI_NSR or GICD_CLRSPI_SR. If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

Accessing GICD_CLRSPI_NSR

Writes to this register have no effect if:

- The value written specifies a Secure SPI, the value is written by a Non-secure access, and the value of the corresponding GICD_NSACR<n> register is less than `0b10`.
- The value written specifies an invalid SPI.
- The SPI is not pending.

16-bit accesses to bits [15:0] of this register must be supported.

Note

A Secure access to this register can clear the pending state of any valid SPI.

GICD_CLRSPI_NSR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x0048	GICD_CLRSPI_NSR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_CLRSPI_SR, Clear Secure SPI Pending Register

The GICD_CLRSPI_SR characteristics are:

Purpose

Removes the pending state from a valid SPI.

A write to this register changes the state of a pending SPI to inactive, and the state of an active and pending SPI to active.

Configuration

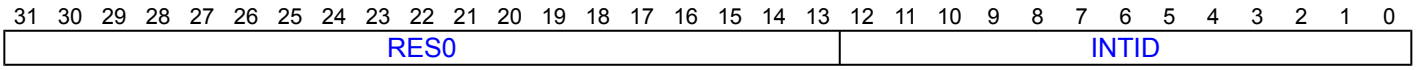
If GICD_TYPER.MBIS == 0, this register is reserved.

When GICD_CTLR.DS == 1, this register is WI.

Attributes

GICD_CLRSPI_SR is a 32-bit register.

Field descriptions



Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

The INTID of the SPI.

Additional information

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to GICD_SETSPI_NSR or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to GICD_CLRSPI_NSR, GICD_CLRSPI_SR, or GICD_ICPENDR<n>.
- For a level-sensitive interrupt, a write to GICD_SETSPI_NSR or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to GICD_CLRSPI_NSR or GICD_CLRSPI_SR. If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

Accessing GICD_CLRSPI_SR

Writes to this register have no effect if:

- The value is written by a Non-secure access.
- The value written specifies an invalid SPI.
- The SPI is not pending.

16-bit accesses to bits [15:0] of this register must be supported.

GICD_CLRSPI_SR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x0058	GICD_CLRSPI_SR

Accessible as follows:

- When GICD_CTLR.DS == 1, accesses to this register are **WI**.
- When GICD_CTLR.DS == 0 and an access is Secure, accesses to this register are **WO**.
- When GICD_CTLR.DS == 0 and an access is Non-secure, accesses to this register are **WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Root, accesses to this register are **WO**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Realm, accesses to this register are **WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_CPENDSGIR<n>, SGI Clear-Pending Registers, n = 0 - 3

The GICD_CPENDSGIR<n> characteristics are:

Purpose

Removes the pending state from an SGI.

A write to this register changes the state of a pending SGI to inactive, and the state of an active and pending SGI to active.

Configuration

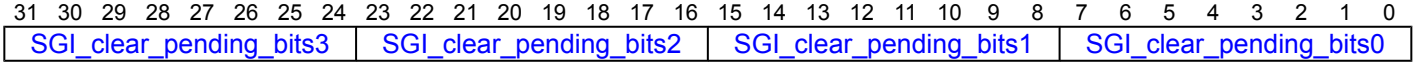
Four SGI clear-pending registers are implemented. Each register contains eight clear-pending bits for each of four SGIs, for a total of 16 possible SGIs.

In multiprocessor implementations, each PE has a copy of these registers.

Attributes

GICD_CPENDSGIR<n> is a 32-bit register.

Field descriptions



SGI_clear_pending_bits<x>, bits [8x+7:8x], for x = 3 to 0

Removes the pending state from SGI number 4n + x for the PE corresponding to the bit number written to.

Reads and writes have the following behavior:

SGI_clear_pending_bits<x>	Meaning
`0x00`	If read, indicates that the SGI from the corresponding PE is not pending and is not active and pending. If written, has no effect.
`0x01`	If read, indicates that the SGI from the corresponding PE is pending or is active and pending. If written, removes the pending state from the SGI for the corresponding PE.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'00000000'`.

Additional information

For SGI ID m, generated by processing element C writing to the corresponding GICD_SGIR field, where DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_CPENDSGIR<n> number is given by n = m DIV 4.
- The offset of the required register is (`0xF10` + (4n)).
- The offset of the required field within the register GICD_CPENDSGIR<n> is given by m MOD 4.
- The required bit in the 8-bit SGI clear-pending field m is bit C.

Accessing GICD_CPENDSGIR<n>

These registers are used only when affinity routing is not enabled. When affinity routing is enabled, this register is RES0. An implementation is permitted to make the register RAZ/WI in this case.

A register bit that corresponds to an unimplemented SGI is RAZ/WI.

These registers are byte-accessible.

If the GIC implementation supports two Security states:

- A register bit that corresponds to a Group 0 interrupt is RAZ/WI to Non-secure accesses.
- Register bits corresponding to unimplemented PEs are RAZ/WI.

GICD_CPENDSGIR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0F10` + (4 * n)	GICD_CPENDSGIR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_CTLR, Distributor Control Register

The GICD_CTLR characteristics are:

Purpose

Enables interrupts and affinity routing.

Configuration

The format of this register depends on the Security state of the access and the number of Security states supported, which is specified by GICD_CTLR.DS.

Attributes

GICD_CTLR is a 32-bit register.

Field descriptions

When access is Secure, in a system that supports two Security states:

31	30:8	7	6	5	4	3	2	1	0
RWP	RES0	E1NWF	DS	ARE_NS	ARE_S	RES0	EnableGrp1S	EnableGrp1NS	EnableGrp0

RWP, bit [31]

Register Write Pending. Read only. Indicates whether a register write is in progress or not:

RWP	Meaning
0b0	No register write in progress. The effects of previous register writes to the affected register fields are visible to all logical components of the GIC architecture, including the CPU interfaces.
0b1	Register write in progress. The effects of previous register writes to the affected register fields are not guaranteed to be visible to all logical components of the GIC architecture, including the CPU interfaces, as the effects of the changes are still being propagated.

This field tracks writes to:

- GICD_CTLR[2:0], the Group Enables, for transitions from 1 to 0 only.
- GICD_CTLR[7:4], the ARE bits, E1NWF bit and DS bit.
- GICD_ICENABLER<n>.

Updates to other register fields are not tracked by this field.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [30:8]

Reserved, RES0.

E1NWF, bit [7]

Enable 1 of N Wakeup Functionality.

It is IMPLEMENTATION DEFINED whether this bit is programmable, or RAZ/WI.

If it is implemented, then it has the following behavior:

E1NWF	Meaning
0b0	A PE that is asleep cannot be picked for 1 of N interrupts.
0b1	A PE that is asleep can be picked for 1 of N interrupts as determined by IMPLEMENTATION DEFINED controls.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

DS, bit [6]

Disable Security.

DS	Meaning
0b0	Non-secure accesses are not permitted to access and modify registers that control Group 0 interrupts.
0b1	Non-secure accesses are permitted to access and modify registers that control Group 0 interrupts.

If DS is written from 0 to 1 when GICD_CTLR.ARE_S == 1, then GICD_CTLR.ARE for the single Security state is RAO/WI.

If the Distributor only supports a single Security state, this bit is RAO/WI.

If the Distributor supports two Security states, it IMPLEMENTATION DEFINED whether this bit is programmable or implemented as RAZ/WI.

When this field is set to 1, all accesses to GICD_CTLR access the single Security state view, and all bits are accessible.

When set to 1, this field can only be cleared by a hardware reset.

Writing this bit from 0 to 1 is UNPREDICTABLE if any of the following is true:

- GICD_CTLR.EnableGrp0==1.
- GICD_CTLR.EnableGrp1S==1.
- GICD_CTLR.EnableGrp1NS==1.
- One or more INTID is in the Active or Active and Pending state.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

ARE_NS, bit [5]

Affinity Routing Enable, Non-secure state.

ARE_NS	Meaning
0b0	Affinity routing disabled for Non-secure state.
0b1	Affinity routing enabled for Non-secure state.

When affinity routing is enabled for the Secure state, this field is RAO/WI.

Changing the ARE_NS settings from 0 to 1 is UNPREDICTABLE except when GICD_CTLR.EnableGrp1 Non-secure == 0.

Changing the ARE_NS settings from 1 to 0 is UNPREDICTABLE.

If GICv2 backwards compatibility for Non-secure state is not implemented, this field is RAO/WI.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

ARE_S, bit [4]

Affinity Routing Enable, Secure state.

ARE_S	Meaning
0b0	Affinity routing disabled for Secure state.
0b1	Affinity routing enabled for Secure state.

Changing the ARE_S setting from 0 to 1 is UNPREDICTABLE except when all of the following apply:

- GICD_CTLR.EnableGrp0==0.
- GICD_CTLR.EnableGrp1S==0.
- GICD_CTLR.EnableGrp1NS==0.

Changing the ARE_S settings from 1 to 0 is UNPREDICTABLE.

If GICv2 backwards compatibility for Secure state is not implemented, this field is RAO/WI.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Bit [3]

Reserved, RES0.

EnableGrp1S, bit [2]

Enable Secure Group 1 interrupts.

EnableGrp1S	Meaning
0b0	Secure Group 1 interrupts are disabled.
0b1	Secure Group 1 interrupts are enabled.

If GICD_CTLR.ARE_S == 0, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

EnableGrp1NS, bit [1]

Enable Non-secure Group 1 interrupts.

EnableGrp1NS	Meaning
0b0	Non-secure Group 1 interrupts are disabled.
0b1	Non-secure Group 1 interrupts are enabled.

Note

This field also controls whether LPIs are forwarded to the PE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

EnableGrp0, bit [0]

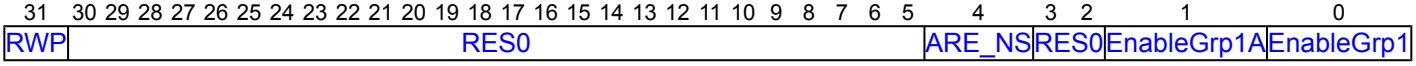
Enable Group 0 interrupts.

EnableGrp0	Meaning
0b0	Group 0 interrupts are disabled.
0b1	Group 0 interrupts are enabled.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

When access is Non-secure, in a system that supports two Security states:



RWP, bit [31]

This bit is a read-only alias of the Secure GICD_CTLR.RWP bit.

Bits [30:5]

Reserved, RES0.

ARE_NS, bit [4]

This bit is a read/write alias of the Secure GICD_CTLR.ARE_NS bit.

If GICv2 backwards compatibility for Non-secure state is not implemented, this field is RAO/WI.

Bits [3:2]

Reserved, RES0.

EnableGrp1A, bit [1]

If ARE_NS == 1, then this bit is a read/write alias of the Secure GICD_CTLR.EnableGrp1NS bit.

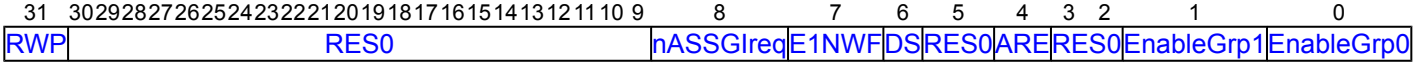
If ARE_NS == 0, then this bit is RES0.

EnableGrp1, bit [0]

If ARE_NS == 0, then this bit is a read/write alias of the Secure GICD_CTLR.EnableGrp1NS bit.

If ARE_NS == 1, then this bit is RES0.

When in a system that supports only a single Security state:



RWP, bit [31]

Register Write Pending. Read only. Indicates whether a register write is in progress or not:

RWP	Meaning
0b0	No register write in progress. The effects of previous register writes to the affected register fields are visible to all logical components of the GIC architecture, including the CPU interfaces.
0b1	Register write in progress. The effects of previous register writes to the affected register fields are not guaranteed to be visible to all logical components of the GIC architecture, including the CPU interfaces, as the effects of the changes are still being propagated.

This field tracks updates to:

- GICD_CTLR[2:0], the Group Enables, for transitions from 1 to 0 only.
- GICD_CTLR[7:4], the ARE bits, E1NWF bit and DS bit.
- GICD_ICENABLER<n>, the bits that allow disabling of SPIs.

Updates to other register fields are not tracked by this field.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [30:9]

Reserved, RES0.

nASSGIreq, bit [8]

When GICv4.1 is implemented:

Controls whether SGIs have an active state.

This bit is RES0 if GICD_TYPER2.GICD_TYPER2.nASSGIcap is 0.

This bit is WI when any of GICD_CTLR.{EnableGrp0,EnableGrp1} is 1.

nASSGIreq	Meaning
0b0	SGIs have an active state and must be deactivated.
0b1	SGIs do not have an active state and do not require deactivation.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

E1NWF, bit [7]

Enable 1 of N Wakeup Functionality.

It is IMPLEMENTATION DEFINED whether this bit is programmable, or RAZ/WI.

If it is implemented, then it has the following behavior:

E1NWF	Meaning
0b0	A PE that is asleep cannot be picked for 1 of N interrupts.
0b1	A PE that is asleep can be picked for 1 of N interrupts as determined by IMPLEMENTATION DEFINED controls.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

DS, bit [6]

Disable Security. This field is RAO/WI.

Bit [5]

Reserved, RES0.

ARE, bit [4]

Affinity Routing Enable.

ARE	Meaning
0b0	Affinity routing disabled.
0b1	Affinity routing enabled.

Changing the ARE settings from 0 to 1 is UNPREDICTABLE except when all of the following apply:

- GICD_CTLR.EnableGrp1==0.
- GICD_CTLR.EnableGrp0==0.

Changing ARE from 1 to 0 is UNPREDICTABLE.

If GICv2 backwards compatibility is not implemented, this field is RAO/WI.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'0'`.

Bits [3:2]

Reserved, RES0.

EnableGrp1, bit [1]

Enable Group 1 interrupts.

EnableGrp1	Meaning
`0b0`	Group 1 interrupts disabled.
`0b1`	Group 1 interrupts enabled.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

EnableGrp0, bit [0]

Enable Group 0 interrupts.

EnableGrp0	Meaning
`0b0`	Group 0 interrupts are disabled.
`0b1`	Group 0 interrupts are enabled.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICD_CTLR

If an interrupt is pending within a CPU interface when the corresponding GICD_CTLR.EnableGrpX bit is written from 1 to 0 the interrupt must be retrieved from the CPU interface.

Note

This might have no effect on the forwarded interrupt if it has already been activated. When a write changes the value of ARE for a Security state or the value of the DS bit, the format used for interpreting the remaining bits provided in the write data is the format that applied before the write takes effect.

GICD_CTLR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0000`	GICD_CTLR

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICACTIVER<n>, Interrupt Clear-Active Registers, n = 0 - 31

The GICD_ICACTIVER<n> characteristics are:

Purpose

Deactivates the corresponding interrupt. These registers are used when saving and restoring GIC state.

Configuration

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ICACTIVER<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ICACTIVER0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ICACTIVER0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ICACTIVER<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Clear_active_bit31	Clear_active_bit30	Clear_active_bit29	Clear_active_bit28	Clear_active_bit27	Clear_active_bit26	Clear_active_bit2

Clear_active_bit<x>, bit [x], for x = 31 to 0

Removes the active state from interrupt number 32n + x. Reads and writes have the following behavior:

Clear_active_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICACTIVER<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_ICACTIVER is (0x380 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

Accessing GICD_ICACTIVER<n>

When affinity routing is enabled for the Security state of an interrupt, the bits corresponding to SGIs and PPIs in that Security state are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by GICR_ICACTIVER0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

The effect of a write must be visible in finite time.

GICD_ICACTIVER<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0380` + (4 * n)	GICD_ICACTIVER<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICACTIVER<n>E, Interrupt Clear-Active Registers (extended SPI range), n = 0 - 31

The GICD_ICACTIVER<n>E characteristics are:

Purpose

Removes the active state from the corresponding SPI in the extended SPI range.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_ICACTIVER<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICACTIVER<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_ICACTIVER<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Clear_active_bit31	Clear_active_bit30	Clear_active_bit29	Clear_active_bit28	Clear_active_bit27	Clear_active_bit26	Clear_active_bit2

Clear_active_bit<x>, bit [x], for x = 31 to 0

For the extended SPIs, removes the active state to interrupt number x. Reads and writes have the following behavior:

Clear_active_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICACTIVER<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_ICACTIVER<n>E is (0x1C00 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

Accessing GICD_ICACTIVER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICACTIVER<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

The effect of a write must be visible in finite time.

GICD_ICACTIVER<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x1C00` + (4 * n)	GICD_ICACTIVER<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICENABLER<n>, Interrupt Clear-Enable Registers, n = 0 - 31

The GICD_ICENABLER<n> characteristics are:

Purpose

Disables forwarding of the corresponding interrupt to the CPU interfaces.

Configuration

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ICENABLER<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ICENABLER0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

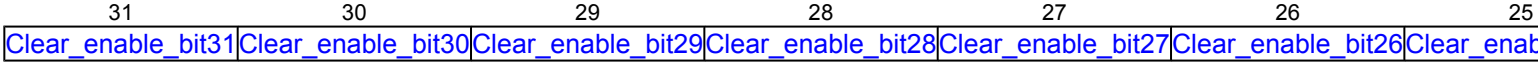
Accessing GICD_ICENABLER0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ICENABLER<n> is a 32-bit register.

Field descriptions



Clear_enable_bit<x>, bit [x], for x = 31 to 0

For SPIs and PPIs, controls the forwarding of interrupt number 32n + x to the CPU interfaces. Reads and writes have the following behavior:

Clear_enable_bit<x>	Meaning
0b0	If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.
0b1	If read, indicates that forwarding of the corresponding interrupt is enabled. If written, disables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0.

For SGIs, the behavior of this bit is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICENABLER<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_ICENABLER is (0x180 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

Note

Writing a 1 to a GICD_ICENABLER<n> bit only disables the forwarding of the corresponding interrupt from the Distributor to any CPU interface. It does not prevent the interrupt from changing state, for example becoming pending or active and pending if it is already active.

Accessing GICD_ICENABLER<n>

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_ICENABLER0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

When GICD_CTLR.DS==0, bits corresponding to Group 0 and Secure Group 1 interrupts are RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether implemented SGIs are permanently enabled, or can be enabled and disabled by writes to GICD_ISENABLER<n> and GICD_ICENABLER<n> where n=0.

Completion of a write to this register does not guarantee that the effects of the write are visible throughout the affinity hierarchy. To ensure an enable has been cleared, software must write to the register with bits set to 1 to clear the required enables. Software must then poll GICD_CTLR.RWP until it has the value zero.

GICD_ICENABLER<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0180` + (4 * n)	GICD_ICENABLER<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICENABLER<n>E, Interrupt Clear-Enable Registers, n = 0 - 31

The GICD_ICENABLER<n>E characteristics are:

Purpose

Disables forwarding of the corresponding SPI in the extended SPI range to the CPU interfaces.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_ICENABLER<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICENABLER<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_ICENABLER<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Clear_enable_bit31	Clear_enable_bit30	Clear_enable_bit29	Clear_enable_bit28	Clear_enable_bit27	Clear_enable_bit26	Clear_enab

Clear_enable_bit<x>, bit [x], for x = 31 to 0

For the extended SPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

Clear_enable_bit<x>	Meaning
0b0	If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.
0b1	If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICENABLER<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_ICENABLER<n>E is (0x1400 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

Accessing GICD_ICENABLER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICENABLER<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ICENABLER<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x1400` + (4 * n)	GICD_ICENABLER<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICFGR<n>, Interrupt Configuration Registers, n = 0 - 63

The GICD_ICFGR<n> characteristics are:

Purpose

Determines whether the corresponding interrupt is edge-triggered or level-sensitive.

Configuration

These registers are available in all GIC configurations. If the GIC implementation supports two Security states, these registers are Common.

GICD_ICFGR1 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ICFGR1 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_ICFGR<n>

For each supported PPI, it is IMPLEMENTATION DEFINED whether software can program the corresponding Int_config field.

For SGIs, Int_config fields are RO, meaning that GICD_ICFGR0 is RO.

Changing Int_config when the interrupt is individually enabled is UNPREDICTABLE.

Changing the interrupt configuration between level-sensitive and edge-triggered (in either direction) at a time when there is a pending interrupt will leave the interrupt in an UNKNOWN pending state.

Fields corresponding to unimplemented interrupts are RAZ/WI.

Attributes

GICD_ICFGR<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
Int_config15		Int_config14		Int_config13		Int_config12		Int_config11		Int_config10		Int_config9		Int_config8		Int_config7		Int_config6		Int_config5	

Int_config<x>, bits [2x+1:2x], for x = 15 to 0

Indicates whether the interrupt is level-sensitive or edge-triggered.

Int_config<x>	Meaning
0b00	Corresponding interrupt is level-sensitive.
0b10	Corresponding interrupt is edge-triggered.

Int_config[0] (bit [2x]) is RES0.

For SGIs, this field always indicates edge-triggered.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICD_ICFGR<n>

For SPIs and PPIs, when GICD_CTLR.DS==0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

GICD_ICFGR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0C00` + (4 * n)	GICD_ICFGR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICFGR<n>E, Interrupt Configuration Registers (Extended SPI Range), n = 0 - 63

The GICD_ICFGR<n>E characteristics are:

Purpose

Determines whether the corresponding SPI in the extended SPI range is edge-triggered or level-sensitive.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_ICFGR<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICFGR<n>E registers is ((GICD_TYPER.ESPI_range+1)*2). Registers are numbered from 0.

Attributes

GICD_ICFGR<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
Int_config15		Int_config14		Int_config13		Int_config12		Int_config11		Int_config10		Int_config9		Int_config8		Int_config7		Int_config6		Int_config5	

Int_config<x>, bits [2x+1:2x], for x = 15 to 0

Indicates whether the interrupt is level-sensitive or edge-triggered.

Int_config[0] (bit[2x]) is RES0.

Int_config<x>	Meaning
0b00	Corresponding interrupt is level-sensitive.
0b10	Corresponding interrupt is edge-triggered.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICD_ICFGR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICFGR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ICFGR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x3000 + (4 * n)	GICD_ICFGR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICPENDR<n>, Interrupt Clear-Pending Registers, n = 0 - 31

The GICD_ICPENDR<n> characteristics are:

Purpose

Removes the pending state from the corresponding interrupt.

Configuration

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ICPENDR<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ICPENDR0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ICPENDR0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ICPENDR<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	
Clear_pending_bit31	Clear_pending_bit30	Clear_pending_bit29	Clear_pending_bit28	Clear_pending_bit27	Clear_pending_bit26	Clea

Clear_pending_bit<x>, bit [x], for x = 31 to 0

For SPIs and PPIs, removes the pending state from interrupt number 32n + x. Reads and writes have the following behavior:

Clear_pending_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not pending on any PE. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases: • If the interrupt is an SGI. In this case, the write is ignored. The pending state of an SGI can be cleared using GICD_CPENDSGIR<n>. • If the interrupt is not pending and is not active and pending. • If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICD_ISPENDR<n>. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICPENDR<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_ICPENDR is (`0x200` + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

Accessing GICD_ICPENDR<n>

Clear-pending bits for SGIs are RO/WI.

When affinity routing is enabled for the Security state of an interrupt:

- Bits corresponding to SGIs and PPIs are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by GICR_ICPENDR0.
- Bits corresponding to Group 0 and Group 1 Secure interrupts can only be cleared by Secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

GICD_ICPENDR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0280` + (4 * n)	GICD_ICPENDR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICPENDR<n>E, Interrupt Clear-Pending Registers (extended SPI range), n = 0 - 31

The GICD_ICPENDR<n>E characteristics are:

Purpose

Removes the pending state to the corresponding SPI in the extended SPI range.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_ICPENDR<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICPENDR<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_ICPENDR<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	
Clear_pending_bit31	Clear_pending_bit30	Clear_pending_bit29	Clear_pending_bit28	Clear_pending_bit27	Clear_pending_bit26	Clea

Clear_pending_bit<x>, bit [x], for x = 31 to 0

For the extended PPIs, removes the pending state to interrupt number x. Reads and writes have the following behavior:

Clear_pending_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases: • If the interrupt is not pending and is not active and pending. • If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICD_ISPENDR<n>E. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICPENDR<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_ICPENDR<n>E is (0x1800 + (4*n)).

- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

Accessing GICD_ICPENDR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICPENDR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ICPENDR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x1800` + (4 * n)	GICD_ICPENDR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IGROUPR<n>, Interrupt Group Registers, n = 0 - 31

The GICD_IGROUPR<n> characteristics are:

Purpose

Controls whether the corresponding interrupt is in Group 0 or Group 1.

Configuration

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Secure.

The number of implemented GICD_IGROUPR<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_IGROUPR0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_IGROUPR0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_IGROUPR<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Group_status_bit31	Group_status_bit30	Group_status_bit29	Group_status_bit28	Group_status_bit27	Group_status_bit26	Group_sta

Group_status_bit<x>, bit [x], for x = 31 to 0

Group status bit.

Group_status_bit<x>	Meaning
0b0	When GICD_CTLR.DS==1, the corresponding interrupt is Group 0. When GICD_CTLR.DS==0, the corresponding interrupt is Secure.
0b1	When GICD_CTLR.DS==1, the corresponding interrupt is Group 1. When GICD_CTLR.DS==0, the corresponding interrupt is Non-secure Group 1.

If affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICD_IGRPMODR<n> to form a 2-bit field that defines an interrupt group. The encoding of this field is described in GICD_IGRPMODR<n>.

If affinity routing is disabled for the Security state of an interrupt, then:

- The corresponding GICD_IGRPMODR<n> bit is RES0.
- For Secure interrupts, the interrupt is Secure Group 0.
- For Non-secure interrupts, the interrupt is Non-secure Group 1.

The reset behavior of this field is:

- On a GIC reset:
  - When n == 0, this field resets to an UNKNOWN value.
  - When n > 0, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGROUP<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_IGROUP is (`0x080` + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

Accessing GICD_IGROUPR<n>

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_IGROUPR0.

When GICD_CTLR.DS==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Accesses to GICD_IGROUPR0 when affinity routing is not enabled for a Security state access the same state as GICR_IGROUPR0, and must update Redistributor state associated with the PE performing the accesses. Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_IGROUPR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0080` + (4 * n)	GICD_IGROUPR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IGROUPR<n>E, Interrupt Group Registers (extended SPI range), n = 0 - 31

The GICD_IGROUPR<n>E characteristics are:

Purpose

Controls whether the corresponding SPI in the extended SPI range is in Group 0 or Group 1.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_IGROUPR<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1:

- The number of implemented GICD_IGROUPR<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.
- When GICD_CTLR.DS==0, this register is Secure.

Attributes

GICD_IGROUPR<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Group_status_bit31	Group_status_bit30	Group_status_bit29	Group_status_bit28	Group_status_bit27	Group_status_bit26	Group_sta

Group_status_bit<x>, bit [x], for x = 31 to 0

Group status bit.

Group_status_bit<x>	Meaning
0b0	When GICD_CTLR.DS==1, the corresponding interrupt is Group 0. When GICD_CTLR.DS==0, the corresponding interrupt is Secure.
0b1	When GICD_CTLR.DS==1, the corresponding interrupt is Group 1. When GICD_CTLR.DS==0, the corresponding interrupt is Non-secure Group 1.

If affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICD_IGRPMODR<n>E to form a 2-bit field that defines an interrupt group. The encoding of this field is described in GICD_IGRPMODR<n>E.

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0:

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGROUPR<n>E number, n, is given by n = (m-4096) DIV 32.

- The offset of the required GICD_IGROUPR<n>E is (`0x1000` + (4*n)).
- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

Accessing GICD_IGROUPR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_IGROUPR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_IGROUPR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x1000` + (4 * n)	GICD_IGROUPR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IGRPMODR<n>, Interrupt Group Modifier Registers, n = 0 - 31

The GICD_IGRPMODR<n> characteristics are:

Purpose

When GICD_CTLR.DS==0, this register together with the GICD_IGROUPR<n> registers, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- Secure Group 1.

Configuration

When GICD_CTLR.DS==0, these registers are Secure.

The number of implemented GICD_IGROUPR<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

When GICD_CTLR.ARE_S==0 or GICD_CTLR.DS==1, the GICD_IGRPMODR<n> registers are RES0. An implementation can make these registers RAZ/WI in this case.

Attributes

GICD_IGRPMODR<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26
Group_modifier_bit31	Group_modifier_bit30	Group_modifier_bit29	Group_modifier_bit28	Group_modifier_bit27	Group_modifier_bit2

Group_modifier_bit<x>, bit [x], for x = 31 to 0

Group modifier bit. When affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICD_IGROUPR<n> to form a 2-bit field that defines an interrupt group:

Group modifier bit	Group status bit	Definition	Short name
0b0	0b0	Secure Group 0	G0S
0b0	0b1	Non-secure Group 1	G1NS
0b1	0b0	Secure Group 1	G1S
0b1	0b1	Reserved, treated as Non-secure Group 1	-

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGRPMODR<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_IGRPMODR is (0x080 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

See GICD_IGROUPR<n> for information about the GICD_IGRPMODR0 reset value.

Accessing GICD_IGRPMODR<n>

When affinity routing is enabled for Secure state, GICD_IGRPMODR0 is RES0 and equivalent functionality is proved by GICR_IGRPMODR0.

When GICD_CTLR.DS==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_IGRPMODR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0D00` + (4 * n)	GICD_IGRPMODR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IGRPMODR<n>E, Interrupt Group Modifier Registers (extended SPI range), n = 0 - 31

The GICD_IGRPMODR<n>E characteristics are:

Purpose

When GICD_CTLR.DS==0, this register together with the GICD_IGROUPR<n>E registers, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- When System register access is enabled, Secure Group 1.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_IGRPMODR<n>E are RES0.

GICD_IGRPMODR<n>E resets to `0x00000000`.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1:

- The number of implemented GICD_IGRPMODR<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.
- When GICD_CTLR.DS==0, this register is Secure.

Attributes

GICD_IGRPMODR<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26
Group_modifier_bit31	Group_modifier_bit30	Group_modifier_bit29	Group_modifier_bit28	Group_modifier_bit27	Group_modifier_bit2

Group_modifier_bit<x>, bit [x], for x = 31 to 0

Group modifier bit. In implementations where affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICD_IGROUPR<n>E to form a 2-bit field that defines an interrupt group:

Group modifier bit	Group status bit	Definition	Short name
`0b0`	`0b0`	Secure Group 0	G0S
`0b0`	`0b1`	Non-secure Group 1	G1NS
`0b1`	`0b0`	Secure Group 1	G1S
`0b1`	`0b1`	Reserved, treated as Non-secure Group 1	-

The reset behavior of this field is:

- On a GIC reset, this field resets to `'0'`.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGRPMODR<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_IGRPMODR<n>E is (`0x3400` + (4*n)).

- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

Accessing GICD_IGRPMODR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_IGRPMODR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_IGRPMODR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x3400` + (4 * n)	GICD_IGRPMODR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IIDR, Distributor Implementer Identification Register

The GICD_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the Distributor.

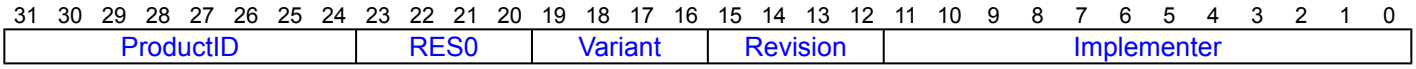
Configuration

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

Attributes

GICD_IIDR is a 32-bit register.

Field descriptions



ProductID, bits [31:24]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

Variant, bits [19:16]

Variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Revision number. Typically, this field is used to distinguish minor revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the Distributor.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for GICD_IIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GICD_IIDR

GICD_IIDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0008`	GICD_IIDR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_INMIR<n>, Non-maskable Interrupt Registers, x = 0 to 31, n = 0 - 31

The GICD_INMIR<n> characteristics are:

Purpose

Holds whether the corresponding SPI has the non-maskable property.

Configuration

This register is present only when GICD_TYPER.NMI == 1. Otherwise, direct accesses to GICD_INMIR<n> are RES0.

The number of implemented GICD_INMIR<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

Attributes

GICD_INMIR<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13
NMI31	NMI30	NMI29	NMI28	NMI27	NMI26	NMI25	NMI24	NMI23	NMI22	NMI21	NMI20	NMI19	NMI18	NMI17	NMI16	NMI15	NMI14	NMI13

NMI<x>, bit [x], for x = 31 to 0

Non-maskable property.

NMI<x>	Meaning
0b0	Interrupt does not have the non-maskable property.
0b1	Interrupt has the non-maskable property.

This bit is RES0 when the corresponding interrupt is configured as Group 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_INMI<n> number, n, is given by n = (m DIV 32).
- The offset of the required GICD_INMI is (0xF80 + (4*n)).
- The bit number of the required in this register is (m MOD 32).

Accessing GICD_INMIR<n>

For SGIs and PPIs:

- The field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_INMIR0.

When affinity routing is not enabled for the Security state of an interrupt in GICD_IGROUPR<n>E, the corresponding bit is RES0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

When GICD_CTLR.DS==0, bits corresponding to Group 0 and Secure Group 1 interrupts are RAZ/WI to Non-secure accesses.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_INMIR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0F80` + (4 * n)	GICD_INMIR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_INMIR<n>E, Non-maskable Interrupt Registers for Extended SPIs, x = 0 to 31, n = 0 - 31

The GICD_INMIR<n>E characteristics are:

Purpose

Holds whether the corresponding SPI in the extended SPI range has the non-maskable property.

Configuration

This register is present only when GICv3.1 is implemented and GICD_TYPER.NMI == 1. Otherwise, direct accesses to GICD_INMIR<n>E are RES0.

When GICD_TYPER.ESPI is 0, these registers are RES0.

When GICD_TYPER.ESPI is 1: the number of implemented GICD_INMIR<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_INMIR<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13
NMI31	NMI30	NMI29	NMI28	NMI27	NMI26	NMI25	NMI24	NMI23	NMI22	NMI21	NMI20	NMI19	NMI18	NMI17	NMI16	NMI15	NMI14	NMI13

NMI<x>, bit [x], for x = 31 to 0

Non-maskable property.

NMI<x>	Meaning
0b0	Interrupt does not have the non-maskable property.
0b1	Interrupt has the non-maskable property.

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0.

This bit is RES0 when the corresponding interrupt is configured as Group 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_INMIR<n>E number, n, is given by n = ((m-4096) DIV 32).
- The offset of the required GICD_INMIR<n>E is (0x3B00 + (4*n)).
- The bit number in this register is ((m-4096) MOD 32).

Accessing GICD_INMIR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_IGROUPR<n>E, the corresponding bit is RES0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

When GICD_CTLR.DS==0, bits corresponding to Group 0 and Secure Group 1 interrupts are RAZ/WI to Non-secure accesses.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_INMIR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x3B00` + (4 * n)	GICD_INMIR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IPRIORITYR<n>, Interrupt Priority Registers, n = 0 - 254

The GICD_IPRIORITYR<n> characteristics are:

Purpose

Holds the priority of the corresponding interrupt.

Configuration

These registers are available in all configurations of the GIC. When GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_IPRIORITYR<n> registers is 8*(GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_IPRIORITYR0 to GICD_IPRIORITYR7 are Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

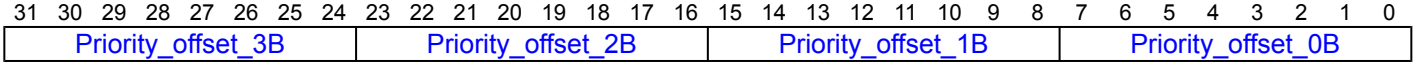
Accessing GICD_IPRIORITYR0 to GICD_IPRIORITYR7 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_IPRIORITYR<n> is a 32-bit register.

Field descriptions



Priority_offset_3B, bits [31:24]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '00000000'.

Priority_offset_2B, bits [23:16]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '00000000'.

Priority_offset_1B, bits [15:8]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '00000000'.

Priority_offset_0B, bits [7:0]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'00000000'`.

Additional information

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IPRIORITYR<n> number, n, is given by n = m DIV 4.
- The offset of the required GICD_IPRIORITYR<n> register is (`0x400` + (4*n)).
- The byte offset of the required Priority field in this register is m MOD 4, where:
  - Byte offset 0 refers to register bits [7:0].
  - Byte offset 1 refers to register bits [15:8].
  - Byte offset 2 refers to register bits [23:16].
  - Byte offset 3 refers to register bits [31:24].

Accessing GICD_IPRIORITYR<n>

These registers are always used when affinity routing is not enabled. When affinity routing is enabled for the Security state of an interrupt:

- GICR_IPRIORITYR<n> is used instead of GICD_IPRIORITYR<n> where n = 0 to 7 (that is, for SGIs and PPIs).
- GICD_IPRIORITYR<n> is RAZ/WI where n = 0 to 7.

These registers are byte-accessible.

A register field corresponding to an unimplemented interrupt is RAZ/WI.

A GIC might implement fewer than eight priority bits, but must implement at least bits [7:4] of each field. In each field, unimplemented bits are RAZ/WI, see 'Interrupt prioritization' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

When GICD_CTLR.DS==0:

- A register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in 'Software accesses of interrupt priority' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

It is IMPLEMENTATION DEFINED whether changing the value of a priority field changes the priority of an active interrupt.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_IPRIORITYR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0400` + (4 * n)	GICD_IPRIORITYR<n>

Accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IPRIORITYR<n>E, Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC., n = 0 - 255

The GICD_IPRIORITYR<n>E characteristics are:

Purpose

Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_IPRIORITYR<n>E are RES0.

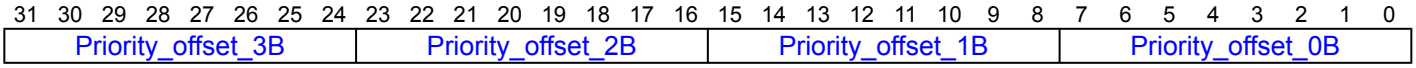
When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_IPRIORITYR<n>E registers is ((GICD_TYPER.ESPI_range+1)*8). Registers are numbered from 0.

Attributes

GICD_IPRIORITYR<n>E is a 32-bit register.

Field descriptions



Priority_offset_3B, bits [31:24]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_2B, bits [23:16]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_1B, bits [15:8]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_0B, bits [7:0]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IPRIORITYR<n> number, n, is given by n = (m-4096) DIV 4.
- The offset of the required GICD_IPRIORITYR<n>E register is (`0x2000` + (4*n)).
- The byte offset of the required Priority field in this register is m MOD 4, where:
  - Byte offset 0 refers to register bits [7:0].
  - Byte offset 1 refers to register bits [15:8].
  - Byte offset 2 refers to register bits [23:16].
  - Byte offset 3 refers to register bits [31:24].

Accessing GICD_IPRIORITYR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISACTIVER<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0:

- A field that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in Software accesses of interrupt priority.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than once. The effect of the change must be visible in finite time.

GICD_IPRIORITYR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x2000` + (4 * n)	GICD_IPRIORITYR<n>E

Accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IROUTER<n>, Interrupt Routing Registers, n = 32 - 1019

The GICD_IROUTER<n> characteristics are:

Purpose

When affinity routing is enabled, provides routing information for the SPI with INTID n.

Configuration

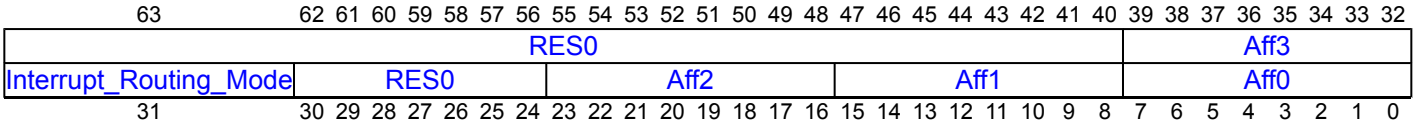
These registers are available in all configurations of the GIC. If the GIC implementation supports two Security states, these registers are Common.

The maximum value of n is given by (32*(GICD_TYPER.ITLinesNumber+1) - 1). GICD_IROUTER<n> registers where n=0 to 31 are reserved.

Attributes

GICD_IROUTER<n> is a 64-bit register.

Field descriptions



Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Interrupt_Routing_Mode, bit [31]

Interrupt Routing Mode. Defines how SPIs are routed in an affinity hierarchy:

Interrupt_Routing_Mode	Meaning
0b0	Interrupts routed to the PE specified by a.b.c.d. In this routing, a, b, c, and d are the values of fields Aff3, Aff2, Aff1, and Aff0 respectively.
0b1	Interrupts routed to any PE defined as a participating node.

If GICD_IROUTER<n>.IRM == 0 and the affinity path does not correspond to an implemented PE, then if the corresponding interrupt becomes pending behavior is CONSTRAINED UNPREDICTABLE:

- The interrupt is not forwarded to any PE, direct reads return the written value
- The affinity path is treated as an UNKNOWN implemented PE, direct reads return the UNKNOWN implemented PE
- The affinity path is treated as an UNKNOWN implemented PE, direct reads return the written value

When GICD_TYPER.No1N is 1, 1 of N distribution is not supported. Setting this field to 1 is CONSTRAINED UNPREDICTABLE, the permitted behaviors are:

- The field behaves as if set to 0 for all purposes.
- The field behaves as if set to 0 for all purposes other than a direct-read of the register.
- The interrupt is treated as not targeting any PE.

When this bit is set to 1, GICD_IROUTER<n>.{Aff3, Aff2, Aff1, Aff0} are UNKNOWN.

Note

An implementation might choose to make the Aff<n> fields RO when this field is 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [30:24]

Reserved, RES0.

Aff2, bits [23:16]

Affinity level 2.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Aff1, bits [15:8]

Affinity level 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Aff0, bits [7:0]

Affinity level 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

For an SPI with INTID m:

- The corresponding GICD_IROUTER<n> register number, n, is given by n = m.
- The offset of the GICD_IROUTER<n> register is `0x6000` + 8n.

Accessing GICD_IROUTER<n>

These registers are used only when affinity routing is enabled. When affinity routing is not enabled:

- These registers are RES0. An implementation is permitted to make the register RAZ/WI in this case.
- The GICD_ITARGETSR<n> registers provide interrupt routing information.

Note

When affinity routing becomes enabled for a Security state (for example, following a reset or following a write to GICD_CTLR) the value of all writable fields in this register is UNKNOWN for

---

that Security state. When the group of an interrupt changes so the ARE setting for the interrupt changes to 1, the value of this register is UNKNOWN for that interrupt.

If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any GICD_IROUTER<n> registers that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

Note

For each interrupt, a GIC implementation might support fewer than 256 values for an affinity level. In this case, some bits of the corresponding affinity level field might be RO. Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_IROUTER<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x6000` + (8 * n)	GICD_IROUTER<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IROUTER<n>E, Interrupt Routing Registers (Extended SPI Range), n = 0 - 1023

The GICD_IROUTER<n>E characteristics are:

Purpose

When affinity routing is enabled, provides routing information for the corresponding SPI in the extended SPI range.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_IROUTER<n>E are RES0.

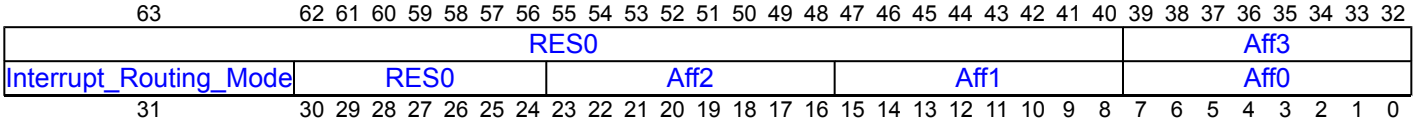
When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_IROUTER<n>E registers is (((GICD_TYPER.ESPI_range+1)*32)-1). Registers are numbered from 0.

Attributes

GICD_IROUTER<n>E is a 64-bit register.

Field descriptions



Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Interrupt_Routing_Mode, bit [31]

Interrupt Routing Mode. Defines how SPIs are routed in an affinity hierarchy:

Interrupt_Routing_Mode	Meaning
0b0	Interrupts routed to the PE specified by a.b.c.d. In this routing, a, b, c, and d are the values of fields Aff3, Aff2, Aff1, and Aff0 respectively.
0b1	Interrupts routed to any PE defined as a participating node.

If GICD_IROUTER<n>E.IRM == 0 and the affinity path does not correspond to an implemented PE, then if the corresponding interrupt becomes pending behavior is CONSTRAINED UNPREDICTABLE:

- The interrupt is not forwarded to any PE, direct reads return the written value
- The affinity path is treated as an UNKNOWN implemented PE, direct reads return the UNKNOWN implemented PE

- The affinity path is treated as an UNKNOWN implemented PE, direct reads return the written value

When GICD_TYPER.No1N is 1, 1 of N distribution is not supported. Setting this field to 1 is CONSTRAINED UNPREDICTABLE, the permitted behaviors are:

- The field behaves as if set to 0 for all purposes.
- The field behaves as if set to 0 for all purposes other than a direct-read of the register.
- The interrupt is treated as not targeting any PE.

When this bit is set to 1, GICD_IROUTER<n>E.{Aff3, Aff2, Aff1, Aff0} are UNKNOWN.

Note

An implementation might choose to make the Aff<n> fields RO when this field is 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [30:24]

Reserved, RES0.

Aff2, bits [23:16]

Affinity level 2.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Aff1, bits [15:8]

Affinity level 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Aff0, bits [7:0]

Affinity level 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

For an SPI with INTID m:

- The corresponding GICD_IROUTER<n>E register number, n, is given by n = m.
- The offset of the GICD_IROUTER<n>E register is `0x6000` + 8n.

Accessing GICD_IROUTER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_IROUTER<n>E, the register is RES0.

When GICD_CTLR.DS==0, a register that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_IROUTER<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x8000 + (8 * n)	GICD_IROUTER<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ISACTIVER<n>, Interrupt Set-Active Registers, n = 0 - 31

The GICD_ISACTIVER<n> characteristics are:

Purpose

Activates the corresponding interrupt. These registers are used when saving and restoring GIC state.

Configuration

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ISACTIVER<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ISACTIVER0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ISACTIVER0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ISACTIVER<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24
Set_active_bit31	Set_active_bit30	Set_active_bit29	Set_active_bit28	Set_active_bit27	Set_active_bit26	Set_active_bit25	Set_active_

Set_active_bit<x>, bit [x], for x = 31 to 0

Adds the active state to interrupt number 32n + x. Reads and writes have the following behavior:

Set_active_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is active, or is active and pending. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISACTIVER<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_ISACTIVER is (0x300 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

Accessing GICD_ISACTIVER<n>

When affinity routing is enabled for the Security state of an interrupt, bits corresponding to SGIs and PPIs are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by GICR_ISACTIVER0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

The bit reads as one if the status of the interrupt is active or active and pending. GICD_ISPENDR<n> and GICD_ICPENDR<n> provide the pending status of the interrupt.

The effect of a write must be visible in finite time. Reading back the written value from either the Set-Active or Clear-Active registers guarantees that a deactivate from a CPU interface Ordered-after the read, will observe the effects of the write on the Active state of the interrupt.

GICD_ISACTIVER<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0300` + (4 * n)	GICD_ISACTIVER<n>

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ISACTIVER<n>E, Interrupt Set-Active Registers (extended SPI range), n = 0 - 31

The GICD_ISACTIVER<n>E characteristics are:

Purpose

Adds the active state to the corresponding SPI in the extended SPI range.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_ISACTIVER<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ISACTIVER<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_ISACTIVER<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24
Set_active_bit31	Set_active_bit30	Set_active_bit29	Set_active_bit28	Set_active_bit27	Set_active_bit26	Set_active_bit25	Set_active_

Set_active_bit<x>, bit [x], for x = 31 to 0

For the extended SPIs, adds the active state to interrupt number x. Reads and writes have the following behavior:

Set_active_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is active, or active and pending on this PE. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISACTIVER<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_ISACTIVER<n>E is (0x1A00 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

Accessing GICD_ISACTIVER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISACTIVER<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

The effect of a write must be visible in finite time. Reading back the written value from either the Set-Active or Clear-Active registers guarantees that a deactivate from a CPU interface Ordered-after the read, will observe the effects of the write on the Active state of the interrupt.

GICD_ISACTIVER<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x1A00` + (4 * n)	GICD_ISACTIVER<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ISENABLER<n>, Interrupt Set-Enable Registers, n = 0 - 31

The GICD_ISENABLER<n> characteristics are:

Purpose

Enables forwarding of the corresponding interrupt to the CPU interfaces.

Configuration

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ISENABLER<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ISENABLER0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ISENABLER0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ISENABLER<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	
Set_enable_bit31	Set_enable_bit30	Set_enable_bit29	Set_enable_bit28	Set_enable_bit27	Set_enable_bit26	Set_enable_bit25	Set_e

Set_enable_bit<x>, bit [x], for x = 31 to 0

For SPIs and PPIs, controls the forwarding of interrupt number 32n + x to the CPU interfaces. Reads and writes have the following behavior:

Set_enable_bit<x>	Meaning
0b0	If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.
0b1	If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1.

For SGIs, the behavior of this bit is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISENABLER<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_ISENABLER is (0x100 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

At start-up, and after a reset, a PE can use this register to discover which peripheral INTIDs the GIC supports. If GICD_CTLR.DS==0 in a system that supports EL3, the PE must do this for the Secure view of the available interrupts, and Non-secure software running on the PE must do this discovery after the Secure software has configured interrupts as Group 0/Secure Group 1 and Non-secure Group 1.

Accessing GICD_ISENABLER<n>

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_ISENABLER0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

When GICD_CTLR.DS==0, bits corresponding to Group 0 or Secure Group 1 interrupts are RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether implemented SGIs are permanently enabled, or can be enabled and disabled by writes to GICD_ISENABLER<n> and GICD_ICENABLER<n> where n=0.

For SPIs and PPIs, each bit controls the forwarding of the corresponding interrupt from the Distributor to the CPU interfaces.

GICD_ISENABLER<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0100` + (4 * n)	GICD_ISENABLER<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ISENABLER<n>E, Interrupt Set-Enable Registers, n = 0 - 31

The GICD_ISENABLER<n>E characteristics are:

Purpose

Enables forwarding of the corresponding SPI in the extended SPI range to the CPU interfaces.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_ISENABLER<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ISENABLER<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_ISENABLER<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	
Set_enable_bit31	Set_enable_bit30	Set_enable_bit29	Set_enable_bit28	Set_enable_bit27	Set_enable_bit26	Set_enable_bit25	Set_e

Set_enable_bit<x>, bit [x], for x = 31 to 0

For the extended SPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

Set_enable_bit<x>	Meaning
0b0	If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.
0b1	If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISENABLER<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_ISENABLER<n>E is (0x1200 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

Accessing GICD_ISENABLER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISENABLER<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ISENABLER<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x1200` + (4 * n)	GICD_ISENABLER<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ISPENDR<n>, Interrupt Set-Pending Registers, n = 0 - 31

The GICD_ISPENDR<n> characteristics are:

Purpose

Adds the pending state to the corresponding interrupt.

Configuration

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ISPENDR<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ISPENDR0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ISPENDR0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ISPENDR<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Set_pending_bit31	Set_pending_bit30	Set_pending_bit29	Set_pending_bit28	Set_pending_bit27	Set_pending_bit26	Set_pending_bi

Set_pending_bit<x>, bit [x], for x = 31 to 0

For SPIs and PPIs, adds the pending state to interrupt number 32n + x. Reads and writes have the following behavior:

Set_pending_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not pending on any PE. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases: • If the interrupt is an SGI. The pending state of an SGI can be set using GICD_SPENDSGIR<n>. • If the interrupt is not inactive and is not active. • If the interrupt is already pending because of a write to GICD_ISPENDR<n>. • If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Accessing GICD_ISPENDR<n>

Set-pending bits for SGIs are read-only and ignore writes. The Set-pending bits for SGIs are provided as GICD_SPENDSGIR<n>.

When affinity routing is enabled for the Security state of an interrupt:

- Bits corresponding to SGIs and PPIs are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by GICR_ISPENDR0.
- Bits corresponding to Group 0 and Group 1 Secure interrupts can only be set by Secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

GICD_ISPENDR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0200` + (4 * n)	GICD_ISPENDR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ISPENDR<n>E, Interrupt Set-Pending Registers (extended SPI range), n = 0 - 31

The GICD_ISPENDR<n>E characteristics are:

Purpose

Adds the pending state to the corresponding SPI in the extended SPI range.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_ISPENDR<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ISPENDR<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_ISPENDR<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Set_pending_bit31	Set_pending_bit30	Set_pending_bit29	Set_pending_bit28	Set_pending_bit27	Set_pending_bit26	Set_pending_bi

Set_pending_bit<x>, bit [x], for x = 31 to 0

For the extended SPIs, adds the pending state to interrupt number x. Reads and writes have the following behavior:

Set_pending_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases: • If the interrupt is already pending because of a write to GICD_ISPENDR<n>E. • If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISPENDR<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_ISPENDR<n>E is (0x1600 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

Accessing GICD_ISPENDR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISPENDR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ISPENDR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x1600` + (4 * n)	GICD_ISPENDR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ITARGETSR<n>, Interrupt Processor Targets Registers, n = 0 - 254

The GICD_ITARGETSR<n> characteristics are:

Purpose

When affinity routing is not enabled, holds the list of target PEs for the interrupt. That is, it holds the list of CPU interfaces to which the Distributor forwards the interrupt if it is asserted and has sufficient priority.

Configuration

These registers are available in all configurations of the GIC. When GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ITARGETSR<n> registers is 8*(GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ITARGETSR0 to GICD_ITARGETSR7 are Banked for each connected PEwith GICR_TYPER.Processor_Number < 8.

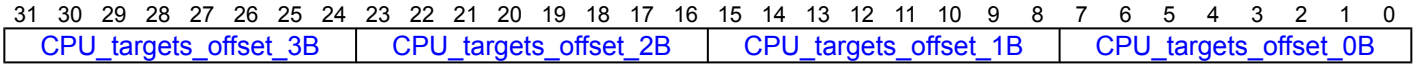
Accessing GICD_ITARGETSR0 to GICD_ITARGETSR7 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ITARGETSR<n> is a 32-bit register.

Field descriptions



PEs in the system number from 0, and each bit in a PE targets field refers to the corresponding PE. For example, a value of `0x3` means that the Pending interrupt is sent to PEs 0 and 1. For GICD_ITARGETSR0-GICD_ITARGETSR7, a read of any targets field returns the number of the PE performing the read.

CPU_targets_offset_3B, bits [31:24]

PE targets for an interrupt, at byte offset 3.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

CPU_targets_offset_2B, bits [23:16]

PE targets for an interrupt, at byte offset 2.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

CPU_targets_offset_1B, bits [15:8]

PE targets for an interrupt, at byte offset 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

CPU_targets_offset_0B, bits [7:0]

PE targets for an interrupt, at byte offset 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

The bits that are set to 1 in the PE targets field determine which PEs are targeted:

Value of PE targets field	Interrupt targets
0bxxxxxxx1	CPU interface 0
0bxxxxxx1x	CPU interface 1
0bxxxxx1xx	CPU interface 2
0bxxxx1xxx	CPU interface 3
0bxxx1xxxx	CPU interface 4
0bxx1xxxxx	CPU interface 5
0bx1xxxxxx	CPU interface 6
0b1xxxxxxx	CPU interface 7

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ITARGETSR<n> number, n, is given by n = m DIV 4.
- The offset of the required GICD_ITARGETSR<n> register is (0x800 + (4*n)).
- The byte offset of the required Priority field in this register is m MOD 4, where:
  - Byte offset 0 refers to register bits [7:0].
  - Byte offset 1 refers to register bits [15:8].
  - Byte offset 2 refers to register bits [23:16].
  - Byte offset 3 refers to register bits [31:24].

Software can write to these registers at any time. Any change to a targets field value:

- Has no effect on any active interrupt. This means that removing a CPU interface from a targets list does not cancel an active state for interrupts on that CPU interface. There is no effect on interrupts that are active and pending until the active status is cleared, at which time it is treated as a pending interrupt.
- Has an effect on any pending interrupts. This means:
  - Enables the CPU interface to be chosen as a target for the pending interrupt using an IMPLEMENTATION DEFINED mechanism.
  - Removing a CPU interface from the target list of a pending interrupt removes the pending state of the interrupt on that CPU interface.

Accessing GICD_ITARGETSR<n>

These registers are used when affinity routing is not enabled. When affinity routing is enabled for the Security state of an interrupt, the target PEs for an interrupt are defined by GICD_IROUTER<n> and the associated byte in GICD_ITARGETSR<n> is RES0. An implementation is permitted to make the byte RAZ/WI in this case.

- These registers are byte-accessible.
- A register field corresponding to an unimplemented interrupt is RAZ/WI.
- A field bit corresponding to an unimplemented CPU interface is RAZ/WI.
- GICD_ITARGETSR0-GICD_ITARGETSR7 are read-only. Each field returns a value that corresponds only to the PE reading the register.
- It is IMPLEMENTATION DEFINED which, if any, SPIs are statically configured in hardware. The field for such an SPI is read-only, and returns a value that indicates the PE targets for the interrupt.
- If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

In a single connected PE implementation, all interrupts target one PE, and these registers are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_ITARGETSR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x0800 + (4 * n)	GICD_ITARGETSR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_NSACR<n>, Non-secure Access Control Registers, n = 0 - 63

The GICD_NSACR<n> characteristics are:

Purpose

Enables Secure software to permit Non-secure software on a particular PE to create and control Group 0 interrupts.

Configuration

The concept of selective enabling of Non-secure access to Group 0 and Secure Group 1 interrupts applies to SGIs and SPIs.

GICD_NSACR0 is a Banked register used for SGIs. A copy is provided for every PE that has a CPU interface and that supports this feature.

Attributes

GICD_NSACR<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13
NS_access15		NS_access14		NS_access13		NS_access12		NS_access11		NS_access10		NS_access9		NS_access8		NS_access7		NS_ac

NS_access<x>, bits [2x+1:2x], for x = 15 to 0

Controls Non-secure access of the interrupt with ID 16n + x.

If the corresponding interrupt does not support configurable Non-secure access, the field is RAZ/WI.

Otherwise, the field is RW and determines the level of Non-secure control permitted if the interrupt is a Secure interrupt. If the interrupt is a Non-secure interrupt, this field is ignored.

The possible values of each 2-bit field are:

NS_access<x>	Meaning
0b00	No Non-secure access is permitted to fields associated with the corresponding interrupt.
0b01	Non-secure read and write access is permitted to set-pending bits in GICD_ISPENDR<n> associated with the corresponding interrupt. A Non-secure write access to GICD_SETSPI_NSR is permitted to set the pending state of the corresponding interrupt. A Non-secure write access to GICD_SGIR is permitted to generate a Secure SGI for the corresponding interrupt. An implementation might also provide read access to clear-pending bits in GICD_ICPENDR<n> associated with the corresponding interrupt.
0b10	As 0b01, but adds Non-secure read and write access permission to fields associated with the corresponding interrupt in the GICD_ICPENDR<n> registers. A Non-secure write access to GICD_CLRSPI_NSR is permitted to clear the pending state of the corresponding interrupt. Also adds Non-secure read access permission to fields associated with the corresponding interrupt in the GICD_ISACTIVER<n> and GICD_ICACTIVER<n> registers.
0b11	For GICD_NSACR0 this encoding is reserved and treated as 10. For all other GICD_NSACR<n> registers this encoding is treated as 0b10, but adds Non-secure read and write access permission to GICD_ITARGETSR<n> and GICD_IROUTER<n> fields associated with the corresponding interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'00'`.

Additional information

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_NSACR<n> number, n, is given by n = m DIV 16.
- The offset of the required GICD_NSACR<n> register is (`0xE00` + (4*n)).

Note

Because each field in this register comprises two bits, GICD_NSACR0 controls access rights to SGI registers, GICD_NSACR1 controls access to PPI registers (and is always RAZ/WI), and all other GICD_NSACR<n> registers control access to SPI registers.

For compatibility with GICv2, writes to GICD_NSACR0 for a particular PE must be coordinated within the Distributor and must update GICR_NSACR for the Redistributor associated with that PE.

Accessing GICD_NSACR<n>

These registers are always used when affinity routing is not enabled. When affinity routing is enabled for the Secure state, GICD_NSACR0 is RES0 and GICR_NSACR provides equivalent functionality for SGIs.

These registers do not support PPIs, therefore GICD_NSACR1 is RAZ/WI.

GICD_NSACR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0E00` + (4 * n)	GICD_NSACR<n>

Accessible as follows:

- When GICD_CTLR.DS == 1, accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0 and an access is Secure, accesses to this register are **RW**.
- When GICD_CTLR.DS == 0 and an access is Non-secure, accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Root, accesses to this register are **RW**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Realm, accesses to this register are **RAZ/WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_NSACR<n>E, Non-secure Access Control Registers, n = 0 - 63

The GICD_NSACR<n>E characteristics are:

Purpose

Enables Secure software to permit Non-secure software on a particular PE to create and control Group 0 interrupts.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICD_NSACR<n>E are RES0.

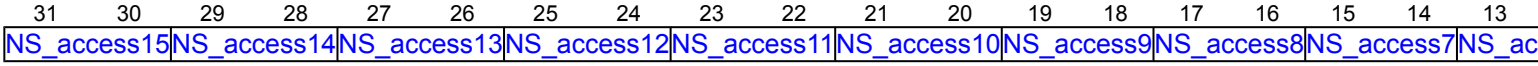
When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICFGR<n>E registers is ((GICD_TYPER.ESPI_range+1)*2). Registers are numbered from 0.

Attributes

GICD_NSACR<n>E is a 32-bit register.

Field descriptions



NS_access<x>, bits [2x+1:2x], for x = 15 to 0

Controls Non-secure access of the interrupt with ID 16n + x.

If the corresponding interrupt does not support configurable Non-secure access, the field is RAZ/WI.

Otherwise, the field is RW and determines the level of Non-secure control permitted if the interrupt is a Secure interrupt. If the interrupt is a Non-secure interrupt, this field is ignored.

The possible values of each 2-bit field are:

NS_access<x>	Meaning
0b00	No Non-secure access is permitted to fields associated with the corresponding interrupt.
0b01	Non-secure read and write access is permitted to set-pending bits in GICD_ISPENDR<n>E associated with the corresponding interrupt. A Non-secure write access to GICD_SETSPI_NSR is permitted to set the pending state of the corresponding interrupt.
0b10	As 0b01, but adds Non-secure read and write access permission to fields associated with the corresponding interrupt in the GICD_ICPENDR<n>E registers. A Non-secure write access to GICD_CLRSPI_NSR is permitted to clear the pending state of the corresponding interrupt. Also adds Non-secure read access permission to fields associated with the corresponding interrupt in the GICD_ISACTIVER<n>E and GICD_ICACTIVER<n>E registers.
0b11	This encoding is treated as 0b10, but adds Non-secure read and write access permission to GICD_IROUTER<n>E fields associated with the corresponding interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to '00'.

Additional information

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_NSACR<n>E number, n, is given by n = (m - 4096) DIV 16.
- The offset of the required GICD_NSACR<n>E register is (`0x3600` + (4*n)).

Accessing GICD_NSACR<n>E

GICD_NSACR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x3600` + (4 * n)	GICD_NSACR<n>E

Accessible as follows:

- When GICD_CTLR.DS == 1, accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0 and an access is Secure, accesses to this register are **RW**.
- When GICD_CTLR.DS == 0 and an access is Non-secure, accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Root, accesses to this register are **RW**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Realm, accesses to this register are **RAZ/WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_SETSPI_NSR, Set Non-secure SPI Pending Register

The GICD_SETSPI_NSR characteristics are:

Purpose

Adds the pending state to a valid SPI if permitted by the Security state of the access and the GICD_NSACR<n> value for that SPI.

A write to this register changes the state of an inactive SPI to pending, and the state of an active SPI to active and pending.

Configuration

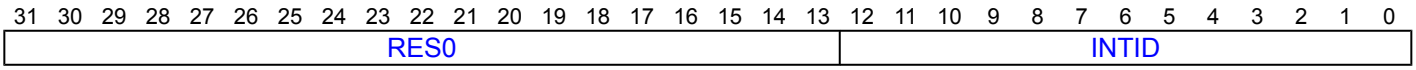
If GICD_TYPER.MBIS == 0, this register is reserved.

When GICD_CTLR.DS == 1, this register provides functionality for all SPIs.

Attributes

GICD_SETSPI_NSR is a 32-bit register.

Field descriptions



Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

The INTID of the SPI.

Additional information

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to GICD_SETSPI_NSR or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to GICD_CLRSPI_NSR, GICD_CLRSPI_SR, or GICD_ICPENDR<n>.
- For a level-sensitive interrupt, a write to GICD_SETSPI_NSR or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to GICD_CLRSPI_NSR or GICD_CLRSPI_SR. If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

Accessing GICD_SETSPI_NSR

Writes to this register have no effect if:

- The value written specifies a Secure SPI, the value is written by a Non-secure access, and the value of the corresponding GICD_NSACR<n> register is 0.
- The value written specifies an invalid SPI.
- The SPI is already pending.

16-bit accesses to bits [15:0] of this register must be supported.

Note

A Secure access to this register can set the pending state of any valid SPI.

GICD_SETSPI_NSR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x0040	GICD_SETSPI_NSR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_SETSPI_SR, Set Secure SPI Pending Register

The GICD_SETSPI_SR characteristics are:

Purpose

Adds the pending state to a valid SPI.

A write to this register changes the state of an inactive SPI to pending, and the state of an active SPI to active and pending.

Configuration

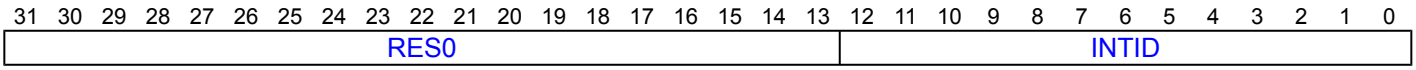
If GICD_TYPER.MBIS == 0, this register is reserved.

When GICD_CTLR.DS == 1, this register is WI.

Attributes

GICD_SETSPI_SR is a 32-bit register.

Field descriptions



Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

The INTID of the SPI.

Additional information

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to GICD_SETSPI_NSR or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to GICD_CLRSPI_NSR, GICD_CLRSPI_SR, or GICD_ICPENDR<n>.
- For a level-sensitive interrupt, a write to GICD_SETSPI_NSR or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to GICD_CLRSPI_NSR or GICD_CLRSPI_SR. If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

Accessing GICD_SETSPI_SR

Writes to this register have no effect if:

- The value is written by a Non-secure access.
- The value written specifies an invalid SPI.
- The SPI is already pending.

16-bit accesses to bits [15:0] of this register must be supported.

GICD_SETSPI_SR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x0050	GICD_SETSPI_SR

Accessible as follows:

- When GICD_CTLR.DS == 1, accesses to this register are **WI**.
- When GICD_CTLR.DS == 0 and an access is Secure, accesses to this register are **WO**.
- When GICD_CTLR.DS == 0 and an access is Non-secure, accesses to this register are **WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Root, accesses to this register are **WO**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Realm, accesses to this register are **WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_SGIR, Software Generated Interrupt Register

The GICD_SGIR characteristics are:

Purpose

Controls the generation of SGIs.

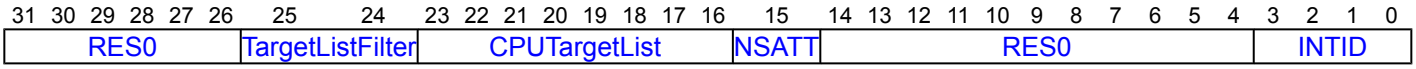
Configuration

This register is available in all configurations of the GIC. If the GIC supports two Security states this register is Common.

Attributes

GICD_SGIR is a 32-bit register.

Field descriptions



Bits [31:26]

Reserved, RES0.

TargetListFilter, bits [25:24]

Determines how the Distributor processes the requested SGI.

TargetListFilter	Meaning
0b00	Forward the interrupt to the CPU interfaces specified by GICD_SGIR.CPUTargetList.
0b01	Forward the interrupt to all CPU interfaces except that of the PE that requested the interrupt.
0b10	Forward the interrupt only to the CPU interface of the PE that requested the interrupt.
0b11	Reserved.

CPUTargetList, bits [23:16]

When GICD_SGIR.TargetListFilter is 0b00, this field defines the CPU interfaces to which the Distributor must forward the interrupt.

Each bit of the field refers to the corresponding CPU interface. For example, CPUTargetList[0] corresponds to interface 0. Setting a bit to 1 indicates that the interrupt must be forwarded to the corresponding interface.

If this field is 0b00000000 when GICD_SGIR.TargetListFilter is 0b00, the Distributor does not forward the interrupt to any CPU interface.

NSATT, bit [15]

Specifies the required group of the SGI.

NSATT	Meaning
0b0	Forward the SGI specified in the INTID field to a specified CPU interface only if the SGI is configured as Group 0 on that interface.
0b1	Forward the SGI specified in the INTID field to a specified CPU interface only if the SGI is configured as Group 1 on that interface.

This field is writable only by a Secure access. Non-secure accesses can also generate Group 0 interrupts, if allowed to do so by GICD_NSACR0. Otherwise, Non-secure writes to GICD_SGIR generate an SGI only if the specified SGI is programmed as Group 1, regardless of the value of bit [15] of the write.

Bits [14:4]

Reserved, RES0.

INTID, bits [3:0]

The INTID of the SGI to forward to the specified CPU interfaces.

Accessing GICD_SGIR

This register is used only when affinity routing is not enabled. When affinity routing is enabled, this register is RES0.

It is IMPLEMENTATION DEFINED whether this register has any effect when the forwarding of interrupts by the Distributor is disabled by GICD_CTLR.

GICD_SGIR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x0F00	GICD_SGIR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_SPENDSGIR<n>, SGI Set-Pending Registers, n = 0 - 3

The GICD_SPENDSGIR<n> characteristics are:

Purpose

Adds the pending state to an SGI.

A write to this register changes the state of an inactive SGI to pending, and the state of an active SGI to active and pending.

Configuration

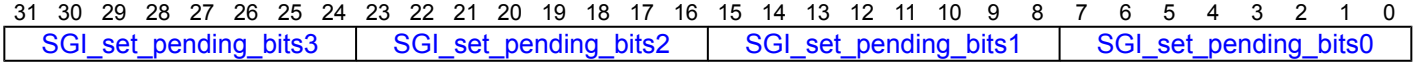
Four SGI set-pending registers are implemented. Each register contains eight set-pending bits for each of four SGIs, for a total of 16 possible SGIs.

In multiprocessor implementations, each PE has a copy of these registers.

Attributes

GICD_SPENDSGIR<n> is a 32-bit register.

Field descriptions



SGI_set_pending_bits<x>, bits [8x+7:8x], for x = 3 to 0

Adds the pending state to SGI number 4n + x for the PE corresponding to the bit number written to.

Reads and writes have the following behavior:

SGI_set_pending_bits<x>	Meaning
0x00	If read, indicates that the SGI from the corresponding PE is not pending and is not active and pending. If written, has no effect.
0x01	If read, indicates that the SGI from the corresponding PE is pending or is active and pending. If written, adds the pending state to the SGI for the corresponding PE.

The reset behavior of this field is:

- On a GIC reset, this field resets to '00000000'.

Additional information

For SGI ID m, generated by processing element C writing to the corresponding GICD_SGIR field, where DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_SPENDSGIR<n> number is given by n = m DIV 4.
- The offset of the required register is (0xF20 + (4n)).
- The offset of the required field within the register GICD_SPENDSGIR<n> is given by m MOD 4.
- The required bit in the 8-bit SGI set-pending field m is bit C.

Accessing GICD_SPENDSGIR<n>

These registers are used only when affinity routing is not enabled. When affinity routing is enabled for the Security state of an interrupt then the bit associated with SGI in that Security state is RES0. An implementation is permitted to make the register RAZ/WI in this case.

A register bit that corresponds to an unimplemented SGI is RAZ/WI.

These registers are byte-accessible.

If the GIC implementation supports two Security states:

- A register bit that corresponds to a Group 0 interrupt is RAZ/WI to Non-secure accesses.
- Register bits corresponding to unimplemented PEs are RAZ/WI.

GICD_SPENDSGIR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	`0x0F20` + (4 * n)	GICD_SPENDSGIR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_STATUSR, Error Reporting Status Register

The GICD_STATUSR characteristics are:

Purpose

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

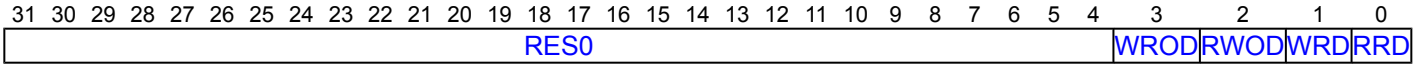
Configuration

If the GIC implementation supports two Security states this register is Banked to provide Secure and Non-secure copies.

Attributes

GICD_STATUSR is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

WROD, bit [3]

Write to an RO location.

WROD	Meaning
0b0	Normal operation.
0b1	A write to an RO location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

RWOD, bit [2]

Read of a WO location.

RWOD	Meaning
0b0	Normal operation.
0b1	A read of a WO location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

WRD, bit [1]

Write to a reserved location.

WRD	Meaning
0b0	Normal operation.
0b1	A write to a reserved location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

RRD	Meaning
0b0	Normal operation.
0b1	A read of a reserved location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

Accessing GICD_STATUSR

This is an optional register. If the register is not implemented, the location is RAZ/WI.

GICD_STATUSR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x0010	GICD_STATUSR (S)

Accessible as follows:

- When an access is Secure, accesses to this register are **RW**.
- When FEAT_RME is implemented and an access is Root, accesses to this register are **RW**.

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x0010	GICD_STATUSR (NS)

Accessible as follows:

- When an access is Non-secure, accesses to this register are **RW**.
- When FEAT_RME is implemented and an access is Realm, accesses to this register are **RAZ/WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_TYPER, Interrupt Controller Type Register

The GICD_TYPER characteristics are:

Purpose

Provides information about what features the GIC implementation supports. It indicates:

- Whether the GIC implementation supports two Security states.
- The maximum number of INTIDs that the GIC implementation supports.
- The number of PEs that can be used as interrupt targets.

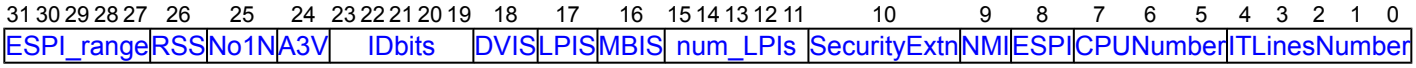
Configuration

This register is available in all configurations of the GIC. When GICD_CTLR.DS==0, this register is Common.

Attributes

GICD_TYPER is a 32-bit register.

Field descriptions



ESPI_range, bits [31:27]

When GICD_TYPER.ESPI == 1:

Indicates the maximum INTID in the Extended SPI range.

Maximum Extended SPI INTID is (32*(ESPI_range + 1) + 4095).

The ESPI_range field only indicates the maximum number of SPIs that the GIC implementation might support. This value determines the number of instances of the following interrupt registers:

- GICD_IGROUPR<n>E.
- GICD_ISENABLER<n>E.
- GICD_ICENABLER<n>E.
- GICD_ISPENDR<n>E.
- GICD_ICPENDR<n>E.
- GICD_ISACTIVER<n>E.
- GICD_ICACTIVER<n>E.
- GICD_IPRIORITYR<n>E.
- GICD_ICFGR<n>E.
- GICD_IROUTER<n>E.
- GICD_IGRPMODR<n>E.

The GIC architecture does not require a GIC implementation to support a continuous range of SPI interrupt IDs. Software must check which SPI INTIDs are supported, up to the maximum value indicated by GICD_TYPER.ESPI_range.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

RSS, bit [26]

Range Selector Support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RSS	Meaning
0b0	The IRI supports targeted SGIs with affinity level 0 values of 0 - 15.
0b1	The IRI supports targeted SGIs with affinity level 0 values of 0 - 255.

Access to this field is **RO**.

No1N, bit [25]

Indicates whether 1 of N SPI interrupts are supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

No1N	Meaning
0b0	1 of N SPI interrupts are supported.
0b1	1 of N SPI interrupts are not supported.

Access to this field is **RO**.

A3V, bit [24]

Affinity 3 valid. Indicates whether the Distributor supports nonzero values of Affinity level 3.

The value of this field is an IMPLEMENTATION DEFINED choice of:

A3V	Meaning
0b0	The Distributor only supports zero values of Affinity level 3.
0b1	The Distributor supports nonzero values of Affinity level 3.

Access to this field is **RO**.

IDbits, bits [23:19]

The number of interrupt identifier bits supported, minus one.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

DVIS, bit [18]

When GICv4 is implemented:

Indicates whether the implementation supports Direct Virtual LPI injection.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DVIS	Meaning
0b0	The implementation does not support Direct Virtual LPI injection.
0b1	The implementation supports Direct Virtual LPI injection.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

LPIS, bit [17]

Indicates whether the implementation supports LPIs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LPIS	Meaning
0b0	The implementation does not support LPIs.
0b1	The implementation supports LPIs.

Access to this field is **RO**.

MBIS, bit [16]

Indicates whether the implementation supports message-based interrupts by writing to Distributor registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MBIS	Meaning
0b0	The implementation does not support message-based interrupts by writing to Distributor registers. The GICD_CLRSPI_NSR, GICD_SETSPI_NSR, GICD_CLRSPI_SR, and GICD_SETSPI_SR registers are reserved.
0b1	The implementation supports message-based interrupts by writing to the GICD_CLRSPI_NSR, GICD_SETSPI_NSR, GICD_CLRSPI_SR, or GICD_SETSPI_SR registers.

Access to this field is **RO**.

num_LPIs, bits [15:11]

Number of supported LPIs.

- 0b00000 Number of LPIs as indicated by GICD_TYPER.IDbits.
- All other values Number of LPIs supported is $2^{(num\_LPIs+1)}$.
  - Available LPI INTIDs are 8192..(8192 + $2^{(num\_LPIs+1)}$ - 1).
  - This field cannot indicate a maximum LPI INTID greater than that indicated by GICD_TYPER.IDbits.

When the supported INTID width is less than 14 bits, this field is RES0 and no LPIs are supported.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SecurityExtn, bit [10]

Indicates whether the GIC implementation supports two Security states:

When GICD_CTLR.DS == 1, this field is RAZ.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SecurityExtn	Meaning
0b0	The GIC implementation supports only a single Security state.
0b1	The GIC implementation supports two Security states.

Access to this field is **RO**.

NMI, bit [9]

Non-maskable Interrupts.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NMI	Meaning
0b0	Non-maskable interrupt property not supported.
0b1	Non-maskable interrupt property is supported.

Access to this field is **RO**.

ESPI, bit [8]

Extended SPI.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ESPI	Meaning
0b0	Extended SPI range not implemented.
0b1	Extended SPI range implemented.

Access to this field is **RO**.

CPUNumber, bits [7:5]

Reports the number of PEs that can be used when affinity routing is not enabled, minus 1.

These PEs must be numbered contiguously from zero, but the relationship between this number and the affinity hierarchy from MPIDR is IMPLEMENTATION DEFINED. If the implementation does not support ARE being zero, this field is 000.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ITLinesNumber, bits [4:0]

For the INTID range 32 to 1019, indicates the maximum SPI supported.

If the value of this field is N, the maximum SPI INTID is 32(N+1) minus 1. For example, 00011 specifies that the maximum SPI INTID is 127.

Regardless of the range of INTIDs defined by this field, interrupt IDs 1020-1023 are reserved for special purposes.

A value of 0 indicates no SPIs are supported.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Additional information

The ITLinesNumber field only indicates the maximum number of SPIs that the GIC implementation might support. This value determines the number of instances of the following interrupt registers:

- GICD_IGROUPR<n>.
- GICD_ISENABLER<n>.
- GICD_ICENABLER<n>.
- GICD_ISPENDR<n>.
- GICD_ICPENDR<n>.
- GICD_ISACTIVER<n>.
- GICD_ICACTIVER<n>.
- GICD_IPRIORITYR<n>.
- GICD_ITARGETSR<n>.
- GICD_ICFGR<n>.
- GICD_IROUTER<n>.
- GICD_IGRPMODR<n>.

The GIC architecture does not require a GIC implementation to support a continuous range of SPI interrupt IDs. Software must check which SPI INTIDs are supported, up to the maximum value indicated by GICD_TYPER.ITLinesNumber.

Accessing GICD_TYPER

GICD_TYPER can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x0004	GICD_TYPER

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_TYPER2, Interrupt Controller Type Register 2

The GICD_TYPER2 characteristics are:

Purpose

Provides information about which features the GIC implementation supports.

Configuration

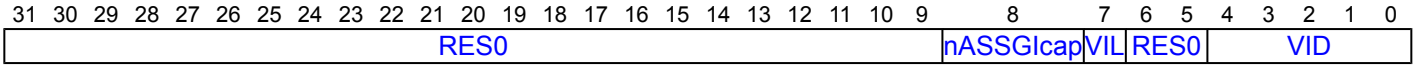
This register is present only when GICv4.1 is implemented. Otherwise, direct accesses to GICD_TYPER2 are RES0.

When GICD_CTLR.DS == 0, this register is Common.

Attributes

GICD_TYPER2 is a 32-bit register.

Field descriptions



Bits [31:9]

Reserved, RES0.

nASSGIcap, bit [8]

Indicates whether SGIs can be configured to not have an active state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

nASSGIcap	Meaning
0b0	SGIs have an active state.
0b1	SGIs can be globally configured not to have an active state.

This bit is RES0 on implementations that support two Security states.

Access to this field is **RO**.

VIL, bit [7]

Indicates whether 16 bits of vPEID are implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VIL	Meaning
0b0	GIC supports 16-bit vPEID.
0b1	GIC supports GICD_TYPER2.VID + 1 bits of vPEID.

Access to this field is **RO**.

Bits [6:5]

Reserved, RES0.

VID, bits [4:0]

When GICD_TYPER2.VIL == 1, the number of bits is equal to the bits of vPEID minus one.

When GICD_TYPER2.VIL == 0, this field is RES0.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GICD_TYPER2

GICD_TYPER2 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	Dist_base	0x000C	GICD_TYPER2

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_APR<n>, Active Priorities Registers, n = 0 - 3

The GICH_APR<n> characteristics are:

Purpose

These registers track which preemption levels are active in the virtual CPU interface, and indicate the current active priority. Corresponding bits are set to 1 in this register when an interrupt is acknowledged, based on GICH_LR<n>.Priority, and the least significant bit set is cleared on EOI.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_APR<n> are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

The number of registers required depends on how many bits are implemented in GICH_LR<n>.Priority:

- When 5 priority bits are implemented, 1 register is required (GICH_APR0).
- When 6 priority bits are implemented, 2 registers are required (GICH_APR0, GICH_APR1).
- When 7 priority bits are implemented, 4 registers are required (GICH_APR0, GICH_APR1, GICH_APR2, GICH_APR3).

Unimplemented registers are RAZ/WI.

Attributes

GICH_APR<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<x>, bit [x], for x = 31 to 0

Active priorities. Possible values of each bit are:

P<x>	Meaning
0b0	There is no interrupt active at the priority corresponding to that bit.
0b1	There is an interrupt active at the priority corresponding to that bit.

The correspondence between priorities and bits depends on the number of bits of priority that are implemented.

If 5 bits of priority are implemented (bits [7:3] of priority), then there are 32 priority groups, and the active state of these priorities are held in GICH_APR0 in the bits corresponding to Priority[7:3].

If 6 bits of priority are implemented (bits [7:2] of priority), then there are 64 priority groups, and:

- The active state of priorities 0 - 124 are held in GICH_APR0 in the bits corresponding to 0:Priority[6:2].
- The active state of priorities 128 - 252 are held in GICH_APR1 in the bits corresponding to 1:Priority[6:2].

If 7 bits of priority are implemented (bits [7:1] of priority), then there are 128 priority groups, and:

- The active state of priorities 0 - 62 are held in GICH_APR0 in the bits corresponding to 00:Priority[5:1].
- The active state of priorities 64 - 126 are held in GICH_APR1 in the bits corresponding to 01:Priority[5:1].
- The active state of priorities 128 - 190 are held in GICH_APR2 in the bits corresponding to 10:Priority[5:1].
- The active state of priorities 192 - 254 are held in GICH_APR3 in the bits corresponding to 11:Priority[5:1].

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing GICH_APR<n>

These registers are used only when System register access is not enabled. When System register access is enabled the following registers provide equivalent functionality:

- In AArch64:
  - For Group 0, ICH_AP0R<n>_EL2.
  - For Group 1, ICH_AP1R<n>_EL2.
- In AArch32:
  - For Group 0, ICH_AP0R<n>.
  - For Group 1, ICH_AP1R<n>.

GICH_APR<n> can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual interface control	`0x00F0` + (4 * n)	GICH_APR<n>

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_EISR, End Interrupt Status Register

The GICH_EISR characteristics are:

Purpose

Indicates which List registers have outstanding EOI maintenance interrupts.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_EISR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_EISR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	
RES0																Status15	Status14	Status13	Status12	Status11	Status10	Status9	Status8	Status7	Status6	Status5	Sta

Bits [31:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

EOI maintenance interrupt status for List register <n>:

Status<n>	Meaning
0b0	GICH_LR<n> does not have an EOI maintenance interrupt.
0b1	GICH_LR<n> has an EOI maintenance interrupt that has not been handled.

For any GICH_LR<n> register, the corresponding status bit is set to 1 if all of the following are true:

- GICH_LR<n>.State is 0b00.
- GICH_LR<n>.HW == 0.
- GICH_LR<n>.EOI == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICH_EISR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICH_EISR provides equivalent functionality.
- For AArch64 implementations, ICH_EISR_EL2 provides equivalent functionality.

Bits corresponding to unimplemented List registers are RAZ.

GICH_EISR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual interface control	0x0020	GICH_EISR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_ELRSR, Empty List Register Status Register

The GICH_ELRSR characteristics are:

Purpose

Indicates which List registers contain valid interrupts.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_ELRSR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_ELRSR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	
RES0																Status15	Status14	Status13	Status12	Status11	Status10	Status9	Status8	Status7	Status6	Status5	Sta

Bits [31:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

Status bit for List register <n>:

Status<n>	Meaning
0b0	GICH_LR<n>, if implemented, contains a valid interrupt. Using this List register can result in overwriting a valid interrupt.
0b1	GICH_LR<n> does not contain a valid interrupt. The List register is empty and can be used without overwriting a valid interrupt or losing an EOI maintenance interrupt.

For any GICH_LR<n> register, the corresponding status bit is set to 1 if GICH_LR<n>.State is 0b00 and either:

- GICH_LR<n>.HW == 1.
- GICH_LR<n>.EOI == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

Accessing GICH_ELRSR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICH_ELRSR provides equivalent functionality.
- For AArch64 implementations, ICH_ELRSR_EL2 provides equivalent functionality.

Bits corresponding to unimplemented List registers are RES0.

GICH_ELRSR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual interface control	0x0030	GICH_ELRSR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_HCR, Hypervisor Control Register

The GICH_HCR characteristics are:

Purpose

Controls the virtual CPU interface.

Configuration

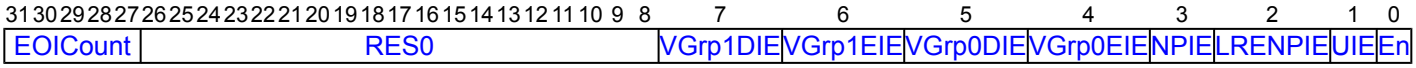
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_HCR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_HCR is a 32-bit register.

Field descriptions



EOICount, bits [31:27]

Counts the number of EOIs received that do not have a corresponding entry in the List registers. The virtual CPU interface increments this field automatically when a matching EOI is received. EOIs that do not clear a bit in GICH_APR<n> do not cause an increment. If an EOI occurs when the value of this field is 31, then the field wraps to 0.

The maintenance interrupt is asserted whenever this field is nonzero and GICH_HCR.LRENPIE == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [26:8]

Reserved, RES0.

VGrp1DIE, bit [7]

VM Group 1 Disabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected virtual machine is disabled:

VGrp1DIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when GICV_CTLR.EnableGrp1 == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VGrp1EIE, bit [6]

VM Group 1 Enabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected virtual machine is enabled:

VGrp1EIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when GICV_CTLR.EnableGrp1 == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VGrp0DIE, bit [5]

VM Group 0 Disabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected virtual machine is disabled:

VGrp0DIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when GICV_CTLR.EnableGrp0 == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VGrp0EIE, bit [4]

VM Group 0 Enabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected virtual machine is enabled:

VGrp0EIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled when GICV_CTLR.EnableGrp0 == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NPIE, bit [3]

No Pending Interrupt Enable.

Enables the signaling of a maintenance interrupt while no pending interrupts are present in the List registers:

NPIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled while the List registers contain no interrupts in the pending state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LRENPIE, bit [2]

List Register Entry Not Present Interrupt Enable.

Enables the signaling of a maintenance interrupt while the virtual CPU interface does not have a corresponding valid List register for an EOI request:

LRENPIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	Maintenance interrupt signaled while GICH_HCR.EOICount is not 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UIE, bit [1]

Underflow Interrupt Enable.

Enables the signaling of a maintenance interrupt when the List registers are either empty or hold only one valid entry.

UIE	Meaning
0b0	Maintenance interrupt disabled.
0b1	A maintenance interrupt is signaled if zero or one of the List register entries are marked as a valid interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

En, bit [0]

Enable.

Global enable bit for the virtual CPU interface.

En	Meaning
0b0	Virtual CPU interface operation is disabled.
0b1	Virtual CPU interface operation is enabled.

When this field is 0:

- The virtual CPU interface does not signal any maintenance interrupts.
- The virtual CPU interface does not signal any virtual interrupts.
- A read of GICV_IAR or GICV_AIAR returns a spurious interrupt ID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information

The VGrp1DIE, VGrp1EIE, VGrp0DIE, and VGrp0EIE fields permit the hypervisor to track the virtual CPU interfaces that are enabled. The hypervisor can then route interrupts that have multiple targets correctly and efficiently, without having to read the virtual CPU interface status.

See 'Maintenance interrupts' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069) and GICH_MISR for more information.

Accessing GICH_HCR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICH_HCR provides equivalent functionality.
- For AArch64 implementations, ICH_HCR_EL2 provides equivalent functionality.

GICH_HCR.En must be set to 1 for any virtual or maintenance interrupt to be asserted.

GICH_HCR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual interface control	0x0000	GICH_HCR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_LR<n>, List Registers, n = 0 - 15

The GICH_LR<n> characteristics are:

Purpose

These registers provide context information for the virtual CPU interface.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_LR<n> are RES0.

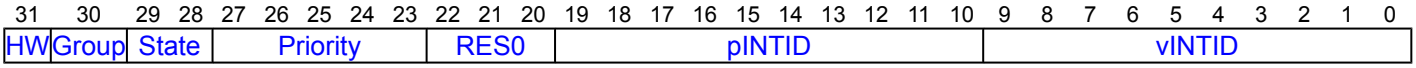
This register is available when the GIC implementation supports interrupt virtualization.

A maximum of 16 List registers can be provided. GICH_VTR.ListRegs defines the number implemented. Unimplemented List registers are RAZ/WI.

Attributes

GICH_LR<n> is a 32-bit register.

Field descriptions



HW, bit [31]

Indicates whether this virtual interrupt is a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt corresponding to the INTID:

HW	Meaning
0b0	This interrupt is triggered entirely in software. No notification is sent to the Distributor when the virtual interrupt is deactivated.
0b1	A hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using GICH_LR<n>.pINTID to indicate the physical interrupt identifier. If GICV_CTLR.EOImode == 0, this request corresponds to a write to GICV_EOIR or GICV_AEOIR, otherwise it corresponds to a write to GICV_DIR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Group, bit [30]

Indicates whether the interrupt is Group 0 or Group 1:

Group	Meaning
0b0	Group 0 virtual interrupt. GICV_CTLR.FIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and GICV_CTLR.EnableGrp0 enables signaling of this interrupt to the virtual machine.
0b1	Group 1 virtual interrupt, signaled as a virtual IRQ. GICV_CTLR.EnableGrp1 enables signaling of this interrupt to the virtual machine.

Note

GICV_CTLR.CBPR controls whether GICV_BPR or GICV_ABPR determines if a pending Group 1 interrupt has sufficient priority to preempt current execution.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

State, bits [29:28]

The state of the interrupt. This field has one of the following values:

State	Meaning
0b00	Inactive
0b01	Pending
0b10	Active
0b11	Active and pending

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. Entries in the inactive state are ignored, except for the purpose of generating virtual maintenance interrupts.

Note

For hardware interrupts, the active and pending state is held in the Distributor rather than the virtual CPU interface. A hypervisor must only use the active and pending state for software originated interrupts, which are typically associated with virtual devices, or for SGIs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Priority, bits [27:23]

The priority of this interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [22:20]

Reserved, RES0.

pINTID, bits [19:10]

The function of this field depends on the value of GICH_LR<n>.HW.

When GICH_LR<n>.HW == 0:

- Bit [19] indicates whether the interrupt triggers an EOI maintenance interrupt. If this bit is 1, then when the interrupt identified by vINTID is deactivated, an EOI maintenance interrupt is asserted.
- Bits [18:13] are reserved, SBZ.
- If the vINTID field value corresponds to an SGI (that is, 0-15), bits [12:10] contain the number of the requesting PE. This appears in the corresponding field of GICV_IAR or GICV_AIAR. If the vINTID field value is not 0-15, this field must be cleared to 0.

When GICH_LR<n>.HW == 1:

- This field indicates the pINTID that the hypervisor forwards to the Distributor. This field is only required to implement enough bits to hold a valid value for the ID configuration. Any unused higher order bits are RAZ/WI.
- If the value of pINTID is 0-15 or 1020-1023, behavior is UNPREDICTABLE. If the value of pINTID is 16-31, this field applies to the PPI associated with this same PE as the virtual CPU interface requesting the deactivation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

vINTID, bits [9:0]

This INTID is returned to the VM when the interrupt is acknowledged through GICV_IAR. Each valid interrupt stored in the List registers must have a unique vINTID for that virtual CPU interface. If the value of vINTID is 1020-1023, behavior is UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICH_LR<n>

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICH_LR<n> provides equivalent functionality.
- For AArch64 implementations, ICH_LR<n>_EL2 provides equivalent functionality.

GICH_LR<n> can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual interface control	`0x0100` + (4 * n)	GICH_LR<n>

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_MISR, Maintenance Interrupt Status Register

The GICH_MISR characteristics are:

Purpose

Indicates which maintenance interrupts are asserted.

Configuration

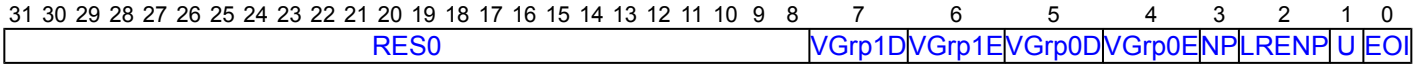
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_MISR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_MISR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

VGrp1D, bit [7]

vPE Group 1 Disabled.

VGrp1D	Meaning
0b0	vPE Group 1 Disabled maintenance interrupt not asserted.
0b1	vPE Group 1 Disabled maintenance interrupt asserted.

This maintenance interrupt is asserted when GICH_HCR.VGrp1DIE == 1 and GICH_VMCR.VENG1 == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp1E, bit [6]

vPE Group 1 Enabled.

VGrp1E	Meaning
0b0	vPE Group 1 Enabled maintenance interrupt not asserted.
0b1	vPE Group 1 Enabled maintenance interrupt asserted.

This maintenance interrupt is asserted when GICH_HCR.VGrp1EIE == 1 and GICH_VMCR.VENG1 == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp0D, bit [5]

vPE Group 0 Disabled.

VGrp0D	Meaning
0b0	vPE Group 0 Disabled maintenance interrupt not asserted.
0b1	vPE Group 0 Disabled maintenance interrupt asserted.

This maintenance interrupt is asserted when GICH_HCR.VGrp0DIE == 1 and GICH_VMCR.VENG0 == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

VGrp0E, bit [4]

vPE Group 0 Enabled.

VGrp0E	Meaning
0b0	vPE Group 0 Enabled maintenance interrupt not asserted.
0b1	vPE Group 0 Enabled maintenance interrupt asserted.

This maintenance interrupt is asserted when GICH_HCR.VGrp0EIE == 1 and GICH_VMCR.VENG0 == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

NP, bit [3]

No Pending.

NP	Meaning
0b0	No Pending maintenance interrupt not asserted.
0b1	No Pending maintenance interrupt asserted.

This maintenance interrupt is asserted when GICH_HCR.NPIE == 1 and no List register is in the pending state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

LRENP, bit [2]

List Register Entry Not Present.

LRENP	Meaning
0b0	List Register Entry Not Present maintenance interrupt not asserted.
0b1	List Register Entry Not Present maintenance interrupt asserted.

This maintenance interrupt is asserted when GICH_HCR.LRENPIE == 1 and GICH_HCR.EOICount is nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

U, bit [1]

Underflow.

U	Meaning
0b0	Underflow maintenance interrupt not asserted.
0b1	Underflow maintenance interrupt asserted.

This maintenance interrupt is asserted when GICH_HCR.UIE == 1 and zero or one of the List register entries are marked as a valid interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

EOI, bit [0]

End Of Interrupt.

EOI	Meaning
0b0	End Of Interrupt maintenance interrupt not asserted.
0b1	End Of Interrupt maintenance interrupt asserted.

This maintenance interrupt is asserted when at least one bit in GICH_EISR == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Additional information

Note

A List register is in the pending state only if the corresponding GICH_LR<n> value is 0b01, that is, pending. The active and pending state is not included.

Accessing GICH_MISR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICH_MISR provides equivalent functionality.
- For AArch64 implementations, ICH_MISR_EL2 provides equivalent functionality.

A maintenance interrupt is asserted only if at least one bit is set to 1 in this register and if GICH_HCR.En == 1.

GICH_MISR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual interface control	0x0010	GICH_MISR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_VMCR, Virtual Machine Control Register

The GICH_VMCR characteristics are:

Purpose

Enables the hypervisor to save and restore the virtual machine view of the GIC state. This register is updated when a virtual machine updates the virtual CPU interface registers.

Configuration

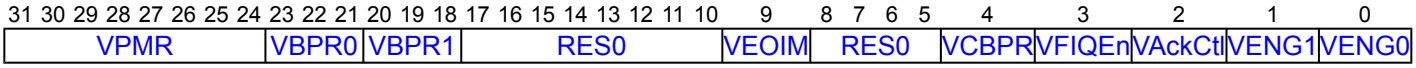
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_VMCR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_VMCR is a 32-bit register.

Field descriptions



VPMR, bits [31:24]

Virtual priority mask. The priority mask level for the CPU interface. If the priority of an interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

This alias field is updated when a VM updates GICV_PMR.Priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VBPR0, bits [23:21]

Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the Group priority field and the subpriority field. The Group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if GICH_VMCR.VCBPR == 1.

This alias field is updated when a VM updates GICV_BPR.Binary_Point.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VBPR1, bits [20:18]

Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the Group priority field and the subpriority field. The Group priority field determines Group 1 interrupt preemption if GICH_VMCR.VCBPR == 0.

This alias field is updated when a VM updates GICV_ABPR.Binary_Point.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [17:10]

Reserved, RES0.

VEOIM, bit [9]

Virtual EOImode. Possible values of this bit are:

VEOIM	Meaning
0b0	A write of an INTID to GICV_EOIR or GICV_AEOIR drops the priority of the interrupt with that INTID, and also deactivates that interrupt.
0b1	A write of an INTID to GICV_EOIR or GICV_AEOIR only drops the priority of the interrupt with that INTID. Software must write to GICV_DIR to deactivate the interrupt.

This alias field is updated when a VM updates GICV_CTLR.EOImode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:5]

Reserved, RES0.

VCBPR, bit [4]

Virtual Common Binary Point Register. Possible values of this bit are:

VCBPR	Meaning
0b0	GICV_ABPR determines the preemption group for Group 1 interrupts.
0b1	GICV_BPR determines the preemption group for Group 1 interrupts.

This alias field is updated when a VM updates GICV_CTLR.CBPR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VFIQEn, bit [3]

Virtual FIQ enable. Possible values of this bit are:

VFIQEn	Meaning
0b0	Group 0 virtual interrupts are presented as virtual IRQs.
0b1	Group 0 virtual interrupts are presented as virtual FIQs.

This alias field is updated when a VM updates GICV_CTLR.FIQEn.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VAckCtl, bit [2]

Virtual AckCtl. Possible values of this bit are:

VAckCtl	Meaning
0b0	If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an INTID of 1022.
0b1	If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the INTID of the corresponding interrupt.

This alias field is updated when a VM updates GICV_CTLR.AckCtl.

This field is supported for backwards compatibility with GICv2. Arm deprecates the use of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VENG1, bit [1]

Virtual interrupt enable, Group 1. Possible values of this bit are:

VENG1	Meaning
0b0	Group 1 virtual interrupts are disabled.
0b1	Group 1 virtual interrupts are enabled.

This alias field is updated when a VM updates GICV_CTLR.EnableGrp1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VENG0, bit [0]

Virtual interrupt enable, Group 0. Possible values of this bit are:

VENG0	Meaning
0b0	Group 0 virtual interrupts are disabled.
0b1	Group 0 virtual interrupts are enabled.

This alias field is updated when a VM updates GICV_CTLR.EnableGrp0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information

Note

A List register is in the pending state only if the corresponding GICH_LR<n> value is 0b01, that is, pending. The active and pending state is not included.

Accessing GICH_VMCR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICH_VMCR provides equivalent functionality.
- For AArch64 implementations, ICH_VMCR_EL2 provides equivalent functionality.

GICH_VMCR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual interface control	0x0008	GICH_VMCR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_VTR, Virtual Type Register

The GICH_VTR characteristics are:

Purpose

Indicates the number of implemented virtual priority bits and List registers.

Configuration

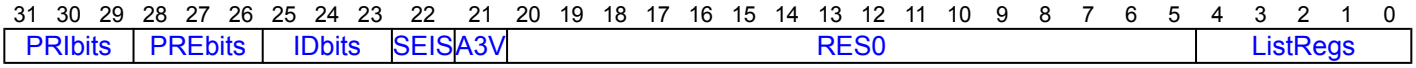
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_VTR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_VTR is a 32-bit register.

Field descriptions



PRIbits, bits [31:29]

The number of virtual priority bits implemented, minus one.

An implementation must implement at least 32 levels of virtual priority (5 priority bits).

PREbits, bits [28:26]

The number of virtual preemption bits implemented, minus one.

An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).

The value of this field must be less than or equal to the value of GICH_VTR.PRIbits.

IDbits, bits [25:23]

The number of virtual interrupt identifier bits supported:

IDbits	Meaning
0b000	16 bits.
0b001	24 bits.

All other values are reserved.

SEIS, bit [22]

SEI support. Indicates whether the virtual CPU interface supports generation of SEIs:

SEIS	Meaning
0b0	The virtual CPU interface logic does not support generation of SEIs.
0b1	The virtual CPU interface logic supports generation of SEIs.

A3V, bit [21]

Affinity 3 valid. Possible values are:

A3V	Meaning
0b0	The virtual CPU interface logic only supports zero values of the Aff3 field in ICC_SGI0R_EL1, ICC_SGI1R_EL1, and ICC_ASGI1R_EL1.
0b1	The virtual CPU interface logic supports nonzero values of the Aff3 field in ICC_SGI0R_EL1, ICC_SGI1R_EL1, and ICC_ASGI1R_EL1.

Bits [20:5]

Reserved, RES0.

ListRegs, bits [4:0]

The number of implemented List registers, minus one.

Accessing GICH_VTR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICH_VTR provides equivalent functionality.
- For AArch64 implementations, ICH_VTR_EL2 provides equivalent functionality.

GICH_VTR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual interface control	0x0004	GICH_VTR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_CLRSPI_NSR, Clear Non-secure SPI Pending Register

The GICM_CLRSPI_NSR characteristics are:

Purpose

Removes the pending state from a valid SPI if permitted by the Security state of the access and the GICD_NSACR<n> value for that SPI.

A write to this register changes the state of a pending SPI to inactive, and the state of an active and pending SPI to active.

Configuration

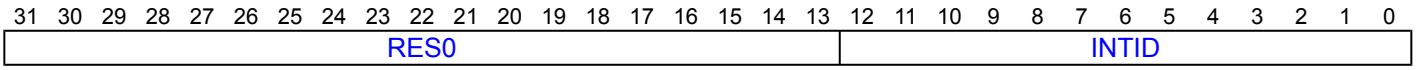
This register is present only when GICM_TYPER.CLR == 1. Otherwise, direct accesses to GICM_CLRSPI_NSR are RES0.

When GICD_CTLR.DS == 1, this register provides functionality for all SPIs.

Attributes

GICM_CLRSPI_NSR is a 32-bit register.

Field descriptions



Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

This field is an alias of GICD_CLRSPI_NSR.

Accessing GICM_CLRSPI_NSR

Writes to this register have no effect if:

- The value written specifies a Secure SPI, the value is written by a Non-secure access, and the value of the corresponding GICD_NSACR<n> register is less than `0b10`.
- The value written specifies an invalid SPI.
- The SPI is not pending.

16-bit accesses to bits [15:0] of this register must be supported.

Note

A Secure access to this register can clear the pending state of any valid SPI.

GICM_CLRSPI_NSR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	MSI_base	`0x0048`	GICM_CLRSPI_NSR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_CLRSPI_SR, Clear Secure SPI Pending Register

The GICM_CLRSPI_SR characteristics are:

Purpose

Removes the pending state from a valid SPI.

A write to this register changes the state of a pending SPI to inactive, and the state of an active and pending SPI to active.

Configuration

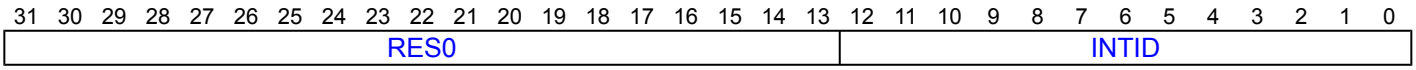
This register is present only when GICM_TYPER.SR == 1 and GICM_TYPER.CLR == 1. Otherwise, direct accesses to GICM_CLRSPI_SR are RES0.

When GICD_CTLR.DS == 1, this register is WI.

Attributes

GICM_CLRSPI_SR is a 32-bit register.

Field descriptions



Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

This field is an alias of GICD_CLRSPI_SR.

Accessing GICM_CLRSPI_SR

Writes to this register have no effect if:

- The value is written by a Non-secure access.
- The value written specifies an invalid SPI.
- The SPI is not pending.

16-bit accesses to bits [15:0] of this register must be supported.

GICM_CLRSPI_SR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	MSI_base	0x0058	GICD_CLRSPI_SR

Accessible as follows:

- When GICD_CTLR.DS == 1, accesses to this register are **WI**.
- When GICD_CTLR.DS == 0 and an access is Secure, accesses to this register are **WO**.
- When GICD_CTLR.DS == 0 and an access is Non-secure, accesses to this register are **WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Root, accesses to this register are **WO**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Realm, accesses to this register are **WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_IIDR, Distributor Implementer Identification Register

The GICM_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the Distributor.

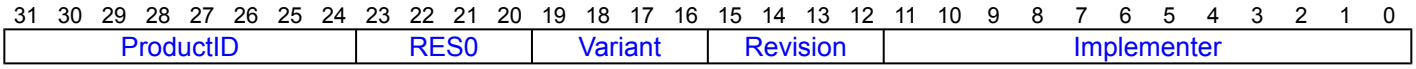
Configuration

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

Attributes

GICM_IIDR is a 32-bit register.

Field descriptions



ProductID, bits [31:24]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

Variant, bits [19:16]

Variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Revision number. Typically, this field is used to distinguish minor revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the Distributor:

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is `0x4`.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore `0x3B`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GICM_IIDR

GICM_IIDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	MSI_base	0x0FCC	GICM_IIDR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_SETSPI_NSR, Set Non-secure SPI Pending Register

The GICM_SETSPI_NSR characteristics are:

Purpose

Adds the pending state to a valid SPI if permitted by the Security state of the access and the GICD_NSACR<n> value for that SPI.

A write to this register changes the state of an inactive SPI to pending, and the state of an active SPI to active and pending.

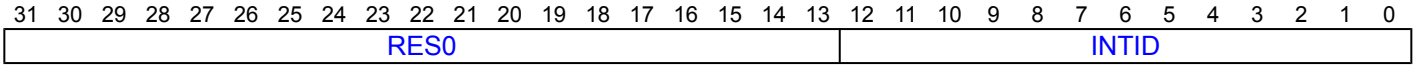
Configuration

When GICD_CTLR.DS==1, this register provides functionality for all SPIs.

Attributes

GICM_SETSPI_NSR is a 32-bit register.

Field descriptions



Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

This field is an alias of GICD_SETSPI_NSR.

Accessing GICM_SETSPI_NSR

Writes to this register have no effect if:

- The value written specifies a Secure SPI, the value is written by a Non-secure access, and the value of the corresponding GICD_NSACR<n> register is 0.
- The value written specifies an invalid SPI.
- The SPI is already pending.

16-bit accesses to bits [15:0] of this register must be supported.

Note

A Secure access to this register can set the pending state of any valid SPI.

GICM_SETSPI_NSR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	MSI_base	0x0040	GICM_SETSPI_NSR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_SETSPI_SR, Set Secure SPI Pending Register

The GICM_SETSPI_SR characteristics are:

Purpose

Adds the pending state to a valid SPI.

A write to this register changes the state of an inactive SPI to pending, and the state of an active SPI to active and pending.

Configuration

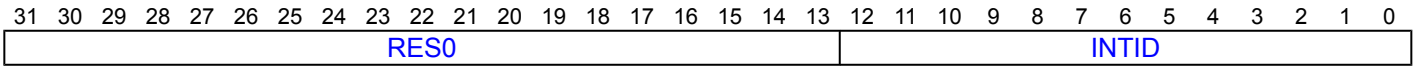
This register is present only when GICM_TYPER.SR == 1. Otherwise, direct accesses to GICM_SETSPI_SR are RES0.

When GICD_CTLR.DS==1, this register is WI.

Attributes

GICM_SETSPI_SR is a 32-bit register.

Field descriptions



Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

This field is an alias of GICD_SETSPI_SR.

Accessing GICM_SETSPI_SR

Writes to this register have no effect if:

- The value is written by a Non-secure access.
- The value written specifies an invalid SPI.
- The SPI is already pending.

16-bit accesses to bits [15:0] of this register must be supported.

GICM_SETSPI_SR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	MSI_base	0x0050	GICM_SETSPI_SR

Accessible as follows:

- When GICD_CTLR.DS == 1, accesses to this register are **WI**.
- When GICD_CTLR.DS == 0 and an access is Secure, accesses to this register are **WO**.
- When GICD_CTLR.DS == 0 and an access is Non-secure, accesses to this register are **WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Root, accesses to this register are **WO**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Realm, accesses to this register are **WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_TYPER, Distributor MSI Type Register

The GICM_TYPER characteristics are:

Purpose

Provides information about what features the GIC implementation supports.

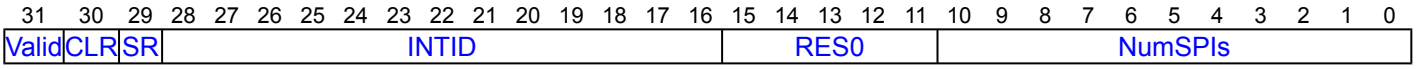
Configuration

This register is available in all configurations of the GIC. When GICD_CTLR.DS==0, this register is Common.

Attributes

GICM_TYPER is a 32-bit register.

Field descriptions



Valid, bit [31]

Reports whether GICM_TYPER content is valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Valid	Meaning
0b0	GICM_TYPER reports no information on the capabilities of the GICM frame, all other fields are RES0.
0b1	GICM_TYPER reports information on capabilities of GICM frame.

Access to this field is **RO**.

CLR, bit [30]

Reports whether MSI clear registers are supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CLR	Meaning
0b0	MSI clear registers not implemented.
0b1	MSI clear registers implemented.

Access to this field is **RO**.

SR, bit [29]

Reports whether Secure aliases of MSI registers are supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SR	Meaning
0b0	Secure aliases of MSI registers not implemented.
0b1	Secure aliases of MSI registers implemented.

Access to this field is **RO**.

INTID, bits [28:16]

INTID of the first SPI assigned to this GICM frame.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [15:11]

Reserved, RES0.

NumSPIs, bits [10:0]

Number of SPIs assigned to this GICM frame.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GICM_TYPER

GICM_TYPER can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Distributor	MSI_base	0x0004	GICM_TYPER

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_CLRLPIR, Clear LPI Pending Register

The GICR_CLRLPIR characteristics are:

Purpose

Clears the pending state of the specified LPI.

Configuration

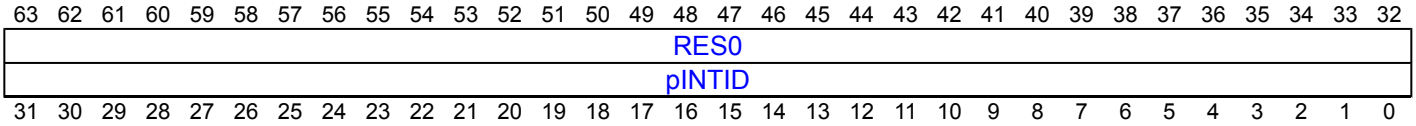
A copy of this register is provided for each Redistributor.

Attributes

GICR_CLRLPIR is a 64-bit register.

Field descriptions

When GICR_TYPER.DirectLPI == 1:



Bits [63:32]

Reserved, RES0.

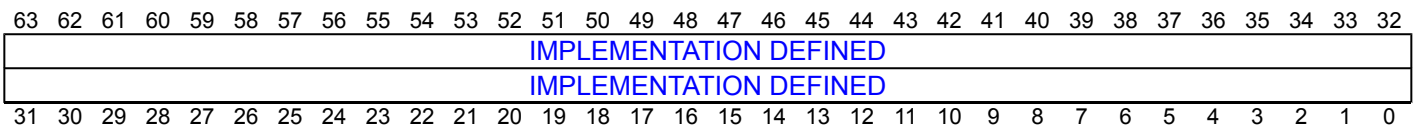
pINTID, bits [31:0]

The INTID of the physical LPI.

Note

The size of this field is IMPLEMENTATION DEFINED, and is specified by the GICD_TYPER.IDbits field. Unimplemented bits are RES0.

When GICR_TYPER.DirectLPI == 0:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing GICR_CLRLPIR

When written with a 32-bit write the data is zero-extended to 64 bits.

This register is mandatory in an implementation that supports LPIs and does not include an ITS. The functionality of this register is IMPLEMENTATION DEFINED in an implementation that does include an ITS.

Writes to this register have no effect if any of the following apply:

- GICR_CTLR.EnableLPIs == 0.
- The pINTID value specifies an unimplemented LPI.
- The pINTID value specifies an LPI that is not pending.

GICR_CLRLPIR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x0048	GICR_CLRLPIR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_CTLR, Redistributor Control Register

The GICR_CTLR characteristics are:

Purpose

Controls the operation of a Redistributor, and enables the signaling of LPIs by the Redistributor to the connected PE.

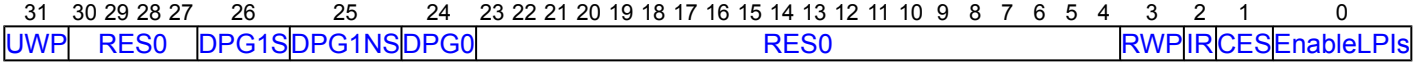
Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_CTLR is a 32-bit register.

Field descriptions



UWP, bit [31]

Upstream Write Pending. Read-only. Indicates whether all upstream writes have been communicated to the Distributor.

UWP	Meaning
0b0	The effects of all upstream writes have been communicated to the Distributor, including any Generate SGI packets. For more information, see 'Generate SGI (ICC)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).
0b1	Not all the effects of upstream writes, including any Generate SGI packets, have been communicated to the Distributor. For more information, see 'Generate SGI (ICC)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Bits [30:27]

Reserved, RES0.

DPG1S, bit [26]

Disable Processor selection for Group 1 Secure interrupts. When GICR_TYPER.DPGS == 1:

DPG1S	Meaning
0b0	A Group 1 Secure SPI configured to use the 1 of N distribution model can select this PE, if the PE is not asleep and if Secure Group 1 interrupts are enabled.
0b1	A Group 1 Secure SPI configured to use the 1 of N distribution model cannot select this PE.

When GICR_TYPER.DPGS == 0 this bit is RAZ/WI.

When GICD_CTLR.DS==1, this field is RAZ/WI. In GIC implementations that support two Security states, this field is only accessible by Secure accesses, and is RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether these bits affect the selection of PEs for interrupts using the 1 of N distribution model when GICD_CTLR.ARE_S==0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

DPG1NS, bit [25]

Disable Processor selection for Group 1 Non-secure interrupts. When GICR_TYPER.DPGS == 1:

DPG1NS	Meaning
0b0	A Group 1 Non-secure SPI configured to use the 1 of N distribution model can select this PE, if the PE is not asleep and if Non-secure Group 1 interrupts are enabled.
0b1	A Group 1 Non-secure SPI configured to use the 1 of N distribution model cannot select this PE.

When GICR_TYPER.DPGS == 0 this bit is RAZ/WI.

It is IMPLEMENTATION DEFINED whether these bits affect the selection of PEs for interrupts using the 1 of N distribution model when GICD_CTLR.ARE_NS==0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

DPG0, bit [24]

Disable Processor selection for Group 0 interrupts. When GICR_TYPER.DPGS == 1:

DPG0	Meaning
0b0	A Group 0 SPI configured to use the 1 of N distribution model can select this PE, if the PE is not asleep and if Group 0 interrupts are enabled.
0b1	A Group 0 SPI configured to use the 1 of N distribution model cannot select this PE.

When GICR_TYPER.DPGS == 0 this bit is RAZ/WI.

When GICD_CTLR.DS == 1, this field is always accessible. In GIC implementations that support two Security states, this field is RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether these bits affect the selection of PEs for interrupts using the 1 of N distribution model when GICD_CTLR.ARE_S == 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Bits [23:4]

Reserved, RES0.

RWP, bit [3]

Register Write Pending. This bit indicates whether a register write for the current Security state is in progress or not.

RWP	Meaning
0b0	The effect of all previous writes to the following registers are visible to all agents in the system: • GICR_ICENABLER0 • GICR_CTLR.DPG1S • GICR_CTLR.DPG1NS • GICR_CTLR.DPG0 • GICR_CTLR, which clears EnableLPIs from 1 to 0. • In FEAT_GICv4p1, GICR_VPROPBASER, which clears Valid from 1 to 0.
0b1	The effect of all previous writes to the following registers are not guaranteed by the architecture to be visible to all agents in the system while the changes are still being propagated: • GICR_ICENABLER0 • GICR_CTLR.DPG1S • GICR_CTLR.DPG1NS • GICR_CTLR.DPG0 • GICR_CTLR, which clears EnableLPIs from 1 to 0. • In FEAT_GICv4p1, GICR_VPROPBASER, which clears Valid from 1 to 0.

IR, bit [2]

LPI invalidate registers supported.

This bit is read-only.

IR	Meaning
0b0	This bit does not indicate whether the GICR_INVLPIR, GICR_INVALLR and GICR_SYNCR are implemented or not.
0b1	GICR_INVLPIR, GICR_INVALLR and GICR_SYNCR are implemented.

If GICR_TYPER.DirectLPI is 1 or GICR_TYPER.RVPEI is 1, GICR_INVLPIR, GICR_INVALLR, and GICR_SYNCR are always implemented.

Arm recommends that implementations report GICR_CTLR.IR as 1 in these cases.

CES, bit [1]

Clear Enable Supported.

This bit is read-only.

CES	Meaning
0b0	The IRI does not indicate whether GICR_CTLR.EnableLPIs is RES1 once set.
0b1	GICR_CTLR.EnableLPIs is not RES1 once set.

Implementing GICR_CTLR.EnableLPIs as programmable and not reporting GICR_CLTR.CES == 1 is deprecated.

Implementing GICR_CTLR.EnableLPIs as RES1 once set is deprecated.

When GICR_CLTR.CES == 0, software cannot assume that GICR_CTLR.EnableLPIs is programmable without observing the bit being cleared to 0.

EnableLPIs, bit [0]

In implementations where affinity routing is enabled for the Security state:

EnableLPIs	Meaning
0b0	LPI support is disabled. Any doorbell interrupt generated as a result of a write to a virtual LPI register must be discarded, and any ITS translation requests or commands involving LPIs in this Redistributor are ignored.
0b1	LPI support is enabled.

Note

If GICR_TYPER.PLPIS == 0, this field is RES0. If GICD_CTLR.ARE_NS is written from 1 to 0 when this bit is 1, behavior is an IMPLEMENTATION DEFINED choice between clearing GICR_CTLR.EnableLPIs to 0 or maintaining its current value.

When affinity routing is not enabled for the Non-secure state, this bit is RES0.

When written from 0 to 1, the Redistributor loads the LPI Pending table from memory to check for any pending interrupts.

After it has been written to 1, it is IMPLEMENTATION DEFINED whether the bit becomes RES1 or can be cleared by to 0.

Where the bit remains programmable:

- Software must observe GICR_CTLR.RWP==0 after clearing GICR_CTLR.EnableLPIs from 1 to 0 before writing GICR_PENDBASER or GICR_PROPBASER, otherwise behavior is UNPREDICTABLE.
- Software must observe GICR_CTLR.RWP==0 after clearing GICR_CTLR.EnableLPIs from 1 to 0 before setting GICR_CTLR.EnableLPIs to 1, otherwise behavior is UNPREDICTABLE.

Note

If one or more ITS is implemented, Arm strongly recommends that all LPIs are mapped to another Redistributor before GICR_CTLR.EnableLPIs is cleared to 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'0'`.

Additional information

The participation of a PE in the 1 of N distribution model for a given interrupt group is governed by the concatenation of GICR_WAKER.ProcessorSleep, the appropriate GICR_CTLR.DPG{1, 0} bit, and the PE interrupt group enable. The behavior options are:

PS	DPG{1S, 1NS, 0}	Enable	PE Behavior
0b0	0b0	0b0	The PE cannot be selected.
0b0	0b0	0b1	The PE can be selected.
0b0	0b1	*	The PE cannot be selected.
0b1	*	*	The PE cannot be selected when GICD_CTLR.E1NWF == 0. When GICD_CTLR.E1NWF == 1, the mechanism by which PEs are selected is IMPLEMENTATION DEFINED.

If an SPI using the 1 of N distribution model has been forwarded to the PE, and a write to GICR_CTLR occurs that changes the DPG bit for the interrupt group of the SPI, the IRI must attempt to select a different target PE for the SPI. This might have no effect on the forwarded SPI if it has already been activated.

Accessing GICR_CTLR

GICR_CTLR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x0000	GICR_CTLR

Accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICACTIVER0, Interrupt Clear-Active Register 0

The GICR_ICACTIVER0 characteristics are:

Purpose

Deactivates the corresponding SGI or PPI. These registers are used when saving and restoring GIC state.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICACTIVER0 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Clear_active_bit31	Clear_active_bit30	Clear_active_bit29	Clear_active_bit28	Clear_active_bit27	Clear_active_bit26	Clear_active_bit2

Clear_active_bit<x>, bit [x], for x = 31 to 0

Removes the active state from interrupt number x. Reads and writes have the following behavior:

Clear_active_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Accessing GICR_ICACTIVER0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICACTIVER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ICACTIVER<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_ICACTIVER<n>.

When GICD_CTLR.DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

The effect of a write must be visible in finite time.

GICR_ICACTIVER0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0380	GICR_ICACTIVER0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICACTIVER<n>E, Interrupt Clear-Active Registers, n = 1 - 2

The GICR_ICACTIVER<n>E characteristics are:

Purpose

Removes the active state from the corresponding PPI.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_ICACTIVER<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICACTIVER<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Clear_active_bit31	Clear_active_bit30	Clear_active_bit29	Clear_active_bit28	Clear_active_bit27	Clear_active_bit26	Clear_active_bit

Clear_active_bit<x>, bit [x], for x = 31 to 0

For the extended PPIs, removes the active state to interrupt number x. Reads and writes have the following behavior:

Clear_active_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ICACTIVER<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_ICACTIVER<n>E is (0x200 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.

Accessing GICR_ICACTIVER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICACTIVER<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

The effect of a write must be visible in finite time.

GICR_ICACTIVER<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	`0x0380` + (4 * n)	GICR_ICACTIVER<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICENABLER0, Interrupt Clear-Enable Register 0

The GICR_ICENABLER0 characteristics are:

Purpose

Disables forwarding of the corresponding SGI or PPI to the CPU interfaces.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICENABLER0 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Clear_enable_bit31	Clear_enable_bit30	Clear_enable_bit29	Clear_enable_bit28	Clear_enable_bit27	Clear_enable_bit26	Clear_enab

Clear_enable_bit<x>, bit [x], for x = 31 to 0

For PPIs and SGIs, controls the forwarding of interrupt number x to the CPU interfaces. Reads and writes have the following behavior:

Clear_enable_bit<x>	Meaning
0b0	If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.
0b1	If read, indicates that forwarding of the corresponding interrupt is enabled. If written, disables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ICENABLER0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICENABLER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ICENABLER<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_ICENABLER<n>.

When GICD_CTLR.DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ICENABLER0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0180	GICR_ICENABLER0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICENABLER<n>E, Interrupt Clear-Enable Registers, n = 1 - 2

The GICR_ICENABLER<n>E characteristics are:

Purpose

Disables forwarding of the corresponding PPI to the CPU interfaces.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_ICENABLER<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICENABLER<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Clear_enable_bit31	Clear_enable_bit30	Clear_enable_bit29	Clear_enable_bit28	Clear_enable_bit27	Clear_enable_bit26	Clear_enab

Clear_enable_bit<x>, bit [x], for x = 31 to 0

For the extended PPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

Clear_enable_bit<x>	Meaning
0b0	If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.
0b1	If read, indicates that forwarding of the corresponding interrupt is enabled. If written, disables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ICENABLER<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_ICENABLER<n>E is (0x180 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.

Accessing GICR_ICENABLER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICENABLER<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ICENABLER<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	`0x0180` + (4 * n)	GICR_ICENABLER<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICFGR0, Interrupt Configuration Register 0

The GICR_ICFGR0 characteristics are:

Purpose

Determines whether the corresponding SGI is edge-triggered or level-sensitive.

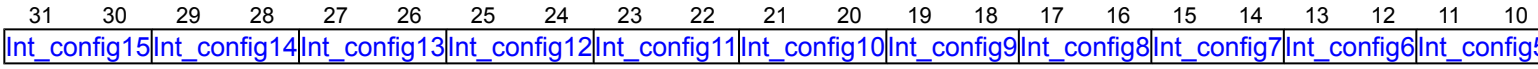
Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICFGR0 is a 32-bit register.

Field descriptions



Int_config<x>, bits [2x+1:2x], for x = 15 to 0

Indicates whether the is level-sensitive or edge-triggered.

Int_config<x>	Meaning
0b00	Corresponding interrupt is level-sensitive.
0b10	Corresponding interrupt is edge-triggered.

SGIs are always edge-triggered.

When the interrupt is visible to the current Security state, a read of this bit always returns the correct value to indicate the interrupt triggering method.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ICFGR0

This register is used when affinity routing is enabled.

When affinity routing is disabled for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case. Equivalent functionality is provided by GICD_ICFGR<n> with n=0.

When GICD_CTLR.DS==0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

GICR_ICFGR0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0C00	GICR_ICFGR0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICFGR1, Interrupt Configuration Register 1

The GICR_ICFGR1 characteristics are:

Purpose

Determines whether the corresponding PPI is edge-triggered or level-sensitive.

Configuration

A copy of this register is provided for each Redistributor.

For each supported PPI, it is IMPLEMENTATION DEFINED whether software can program the corresponding Int_config field.

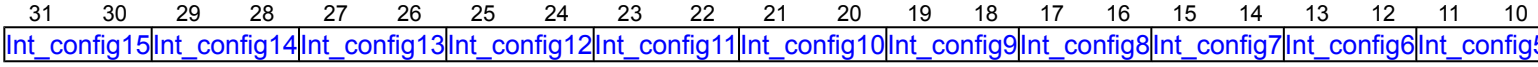
Changing Int_config when the interrupt is individually enabled is UNPREDICTABLE.

Changing the interrupt configuration between level-sensitive and edge-triggered (in either direction) at a time when there is a pending interrupt will leave the interrupt in an UNKNOWN pending state.

Attributes

GICR_ICFGR1 is a 32-bit register.

Field descriptions



Int_config<x>, bits [2x+1:2x], for x = 15 to 0

Indicates whether the interrupt is level-sensitive or edge-triggered.

Int_config<x>	Meaning
0b00	Corresponding interrupt is level-sensitive.
0b10	Corresponding interrupt is edge-triggered.

Int_config[0] (bit [2x]) is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ICFGR1

This register is used when affinity routing is enabled.

When affinity routing is disabled for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case. Equivalent functionality is provided by GICD_ICFGR<n> with n=1.

When GICD_CTLR.DS==0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

GICR_ICFGR1 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0C04	GICR_ICFGR1

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICFGR<n>E, Interrupt configuration registers, n = 2 - 5

The GICR_ICFGR<n>E characteristics are:

Purpose

Determines whether the corresponding PPI in the extended PPI range is edge-triggered or level-sensitive.

Configuration

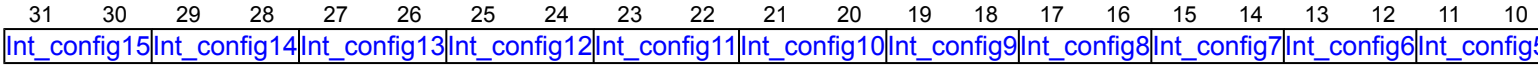
This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_ICFGR<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICFGR<n>E is a 32-bit register.

Field descriptions



Int_config<x>, bits [2x+1:2x], for x = 15 to 0

Indicates whether the interrupt is level-sensitive or edge-triggered.

Int_config[0] (bit [2x]) is RES0.

Int_config<x>	Meaning
0b00	The corresponding interrupt is level-sensitive.
0b10	The corresponding interrupt is edge-triggered.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

For each supported extended PPI, it is IMPLEMENTATION DEFINED whether software can program the corresponding Int_config field.

Accessing GICR_ICFGR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICFGR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ICFGR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0C00 + (4 * n)	GICR_ICFGR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICPENDR0, Interrupt Clear-Pending Register 0

The GICR_ICPENDR0 characteristics are:

Purpose

Removes the pending state from the corresponding SGI or PPI.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICPENDR0 is a 32-bit register.

Field descriptions

31	30	29	28	27	26
Clear_pending_bit31	Clear_pending_bit30	Clear_pending_bit29	Clear_pending_bit28	Clear_pending_bit27	Clear_pending_bit26	Clea

Clear_pending_bit<x>, bit [x], for x = 31 to 0

Removes the pending state from interrupt number x. Reads and writes have the following behavior:

Clear_pending_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases: • If the interrupt is not pending and is not active and pending. • If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICD_ISPENDR<n>. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ICPENDR0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICPENDR0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ICPENDR<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_ICENABLER<n>.

When GICD_CTLR.DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ICPENDR0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0280	GICR_ICPENDR0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICPENDR<n>E, Interrupt Clear-Pending Registers, n = 1 - 2

The GICR_ICPENDR<n>E characteristics are:

Purpose

Removes the pending state from the corresponding PPI.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_ICPENDR<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICPENDR<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26
Clear_pending_bit31	Clear_pending_bit30	Clear_pending_bit29	Clear_pending_bit28	Clear_pending_bit27	Clear_pending_bit26	Clea

Clear_pending_bit<x>, bit [x], for x = 31 to 0

For the extended PPIs, removes the pending state to interrupt number x. Reads and writes have the following behavior:

Clear_pending_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not pending on this PE. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is pending, or active and pending on this PE. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases: • If the interrupt is not pending and is not active and pending. • If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICR_ISPENDR<n>E. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ICPENDR<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_ICPENDR<n>E is (0x200 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.

Accessing GICR_ICPENDR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICPENDR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ICPENDR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	`0x0280` + (4 * n)	GICR_ICPENDR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IGROUPR0, Interrupt Group Register 0

The GICR_IGROUPR0 characteristics are:

Purpose

Controls whether the corresponding SGI or PPI is in Group 0 or Group 1.

Configuration

This register is available in all GIC configurations. If the GIC implementation supports two Security states, this register is Secure.

A copy of this register is provided for each Redistributor.

Attributes

GICR_IGROUPR0 is a 32-bit register.

Field descriptions

31	30	29	28
Redistributor_group_status_bit31	Redistributor_group_status_bit30	Redistributor_group_status_bit29	Redistributor_group_status_b

Redistributor_group_status_bit<x>, bit [x], for x = 31 to 0

Group status bit. In this register:

- Bits [31:16] are group status bits for PPIs.
- Bits [15:0] are group status bits for SGIs.

Redistributor_group_status_bit<x>	Meaning
0b0	When GICD_CTLR.DS==1, the corresponding interrupt is Group 0. When GICD_CTLR.DS==0, the corresponding interrupt is Secure.
0b1	When GICD_CTLR.DS==1, the corresponding interrupt is Group 1. When GICD_CTLR.DS==0, the corresponding interrupt is Non-secure Group 1.

When GICD_CTLR.DS == 0, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICR_IGRPMODR0 to form a 2-bit field that defines an interrupt group. The encoding of this field is at GICR_IGRPMODR0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

The considerations for the reset value of this register are the same as those for GICD_IGROUPR<n> with n=0.

Accessing GICR_IGROUPR0

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGROUPR0, the corresponding bit is RES0 and equivalent functionality is provided by GICD_IGROUPR<n> with n=0.

When GICD_CTLR.DS == 0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICR_IGROUPR0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0080	GICR_IGROUPR0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IGROUPR<n>E, Interrupt Group Registers, n = 1 - 2

The GICR_IGROUPR<n>E characteristics are:

Purpose

Controls whether the corresponding PPI is in Group 0 or Group 1.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_IGROUPR<n>E are RES0.

When GICD_CTLR.DS==0, this register is Secure.

A copy of this register is provided for each Redistributor.

Attributes

GICR_IGROUPR<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Group_status_bit31	Group_status_bit30	Group_status_bit29	Group_status_bit28	Group_status_bit27	Group_status_bit26	Group_sta

Group_status_bit<x>, bit [x], for x = 31 to 0

Group status bit.

Group_status_bit<x>	Meaning
0b0	When GICD_CTLR.DS==1, the corresponding interrupt is Group 0. When GICD_CTLR.DS==0, the corresponding interrupt is Secure.
0b1	When GICD_CTLR.DS==1, the corresponding interrupt is Group 1. When GICD_CTLR.DS==0, the corresponding interrupt is Non-secure Group 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

If affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICR_IGRPMODR<n>E to form a 2-bit field that defines an interrupt group. The encoding of this field is described in GICR_IGRPMODR<n>E.

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_IGROUPR<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_IGROUPR<n>E is (0x080 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.

Accessing GICR_IGROUPR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGROUPR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_IGROUPR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	`0x0080` + (4 * n)	GICR_IGROUPR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IGRPMODR0, Interrupt Group Modifier Register 0

The GICR_IGRPMODR0 characteristics are:

Purpose

When GICD_CTLR.DS==0, this register together with the GICR_IGROUPR0 register, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- When System register access is enabled, Secure Group 1.

Configuration

When GICD_CTLR.DS==0, this register is Secure.

A copy of this register is provided for each Redistributor.

Attributes

GICR_IGRPMODR0 is a 32-bit register.

Field descriptions

31	30	29	28	27	26
Group_modifier_bit31	Group_modifier_bit30	Group_modifier_bit29	Group_modifier_bit28	Group_modifier_bit27	Group_modifier_bit2

Group_modifier_bit<x>, bit [x], for x = 31 to 0

Group modifier bit. In implementations where affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICR_IGROUPR0 to form a 2-bit field that defines an interrupt group:

Group modifier bit	Group status bit	Definition	Short name
0b0	0b0	Secure Group 0	G0S
0b0	0b1	Non-secure Group 1	G1NS
0b1	0b0	Secure Group 1	G1S
0b1	0b1	Reserved, treated as Non-secure Group 1	-

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_IGRPMODR0

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGRPMODR0, the corresponding bit is RES0 and equivalent functionality is provided by GICD_IGRPMODR<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_IGRPMODR<n>.

When GICD_CTLR.ARE_S == 0 or GICD_CTLR.DS == 1, GICR_IGRPMODR0 is RES0. An implementation can make this register RAZ/WI in this case.

When GICD_CTLR.DS==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICR_IGRPMODR0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0D00	GICR_IGRPMODR0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IGRPMODR<n>E, Interrupt Group Modifier Registers, n = 1 - 2

The GICR_IGRPMODR<n>E characteristics are:

Purpose

When GICD_CTLR.DS==0, this register together with the GICR_IGROUPR<n>E registers, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- When System register access is enabled, Secure Group 1.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_IGRPMODR<n>E are RES0.

When GICD_CTLR.DS==0, this register is Secure.

A copy of this register is provided for each Redistributor.

Attributes

GICR_IGRPMODR<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26
Group_modifier_bit31	Group_modifier_bit30	Group_modifier_bit29	Group_modifier_bit28	Group_modifier_bit27	Group_modifier_bit2

Group_modifier_bit<x>, bit [x], for x = 31 to 0

Group modifier bit. In implementations where affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICR_IGROUPR<n>E to form a 2-bit field that defines an interrupt group:

Group modifier bit	Group status bit	Definition	Short name
0b0	0b0	Secure Group 0	G0S
0b0	0b1	Non-secure Group 1	G1NS
0b1	0b0	Secure Group 1	G1S
0b1	0b1	Reserved, treated as Non-secure Group 1	-

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_IGRPMODR<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_IGRPMODR<n>E is (0xD00 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.

Accessing GICR_IGRPMODR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGRPMODR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_IGRPMODR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0D00 + (4 * n)	GICR_IGRPMODR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IIDR, Redistributor Implementer Identification Register

The GICR_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the Redistributor.

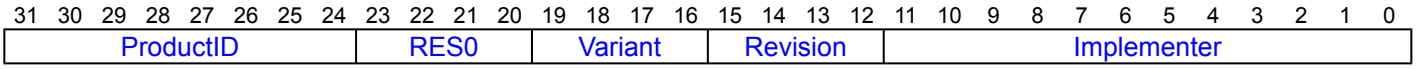
Configuration

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

Attributes

GICR_IIDR is a 32-bit register.

Field descriptions



ProductID, bits [31:24]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

Variant, bits [19:16]

Variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Revision number. Typically, this field is used to distinguish minor revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the Redistributor.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for GICR_IIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GICR_IIDR

GICR_IIDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	`0x0004`	GICR_IIDR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_INMIR0, Non-maskable Interrupt Register 0

The GICR_INMIR0 characteristics are:

Purpose

Controls whether the corresponding SGI or PPI has the non-maskable property.

Configuration

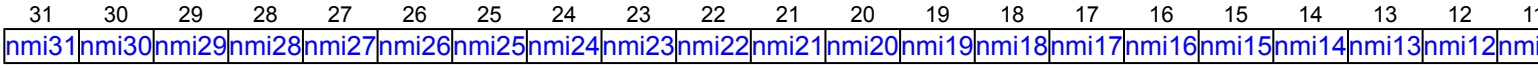
This register is present only when GICD_TYPER.NMI == 1. Otherwise, direct accesses to GICR_INMIR0 are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_INMIR0 is a 32-bit register.

Field descriptions



nmi<x>, bit [x], for x = 31 to 0

Non-maskable property.

nmi<x>	Meaning
0b0	Interrupt does not have the non-maskable property.
0b1	Interrupt has the non-maskable property.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0.

This bit is RES0 when the corresponding interrupt is configured as Group 0.

Accessing GICR_INMIR0

Bits corresponding to unimplemented interrupts are RAZ/WI.

When GICD_CTLR.DS==0, bits corresponding to Group 0 and Secure Group 1 interrupts are RAZ/WI to Non-secure accesses.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICR_INMIR0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0F80	GICR_INMIR0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_INMIR<n>E, Non-maskable Interrupt Registers for Extended PPIs, x = 1 to 2., n = 1 - 2

The GICR_INMIR<n>E characteristics are:

Purpose

Controls whether the corresponding Extended PPI has the non-maskable property.

Configuration

This register is present only when GICv3.1 is implemented and GICD_TYPER.NMI == 1. Otherwise, direct accesses to GICR_INMIR<n>E are RES0.

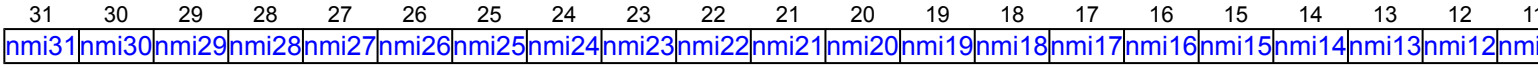
When GICR_TYPER.PPInum is `0b0000`, these registers are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_INMIR<n>E is a 32-bit register.

Field descriptions



nmi<x>, bit [x], for x = 31 to 0

Non-maskable property.

nmi<x>	Meaning
`0b0`	Interrupt does not have the non-maskable property.
`0b1`	Interrupt has the non-maskable property.

This bit is RES0 when the corresponding interrupt is configured as Group 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'0'`.

Additional information

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0.

Accessing GICR_INMIR<n>E

Bits corresponding to unimplemented interrupts are RAZ/WI.

When GICD_CTLR.DS==0, bits corresponding to Group 0 and Secure Group 1 interrupts are RAZ/WI to Non-secure accesses.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICR_INMIR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	`0x0F80` + (4 * n)	GICR_INMIR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_INVALLR, Redistributor Invalidate All Register

The GICR_INVALLR characteristics are:

Purpose

Invalidates any cached configuration data of all LPIs, causing the GIC to reload the interrupt configuration from the appropriate LPI Configuration table.

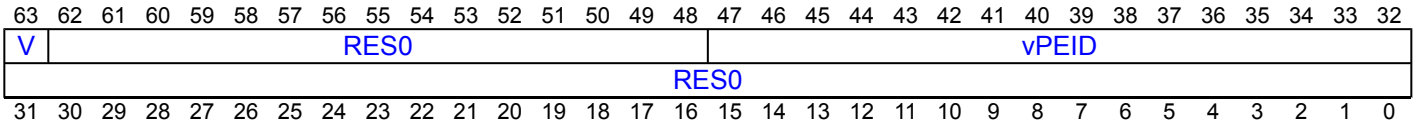
Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_INVALLR is a 64-bit register.

Field descriptions



V, bit [63]

When GICv4.1 is implemented:

Indicates whether the INTID is virtual or physical.

V	Meaning
0b0	Invalidate is for a physical INTID.
0b1	Invalidate is for a virtual INTID.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

vPEID, bits [47:32]

When GICv4.1 is implemented:

When GICR_INVLPIR.V == 0, this field is RES0

When GICR_INVLPIR.V == 1, this field is the target vPEID of the invalidate.

Note

The size of this field is IMPLEMENTATION DEFINED, and is specified by the GICD_TYPER2.VIL and GICD_TYPER2.VID fields. Unimplemented bits are RES0.

Otherwise:

Reserved, RES0.

Bits [31:0]

Reserved, RES0.

Additional information

Note

If any LPI has been forwarded to the PE and a valid write to GICR_INVALLR is received, the Redistributor must ensure it reloads its properties from memory. This has no effect on the forwarded LPI if it has already been activated.

Accessing GICR_INVALLR

This register is mandatory when any of the following are true:

- GICR_TYPER.Direct is 1.
- GICR_CTLR.IR is 1.
- GICv4.1 is implemented.

Otherwise, the functionality is IMPLEMENTATION DEFINED.

Writes to this register have no effect if no physical LPIs are currently stored in the local Redistributor cache.

GICR_INVALLR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x00B0	GICR_INVALLR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_INVLPIR, Redistributor Invalidate LPI Register

The GICR_INVLPIR characteristics are:

Purpose

Invalidates the cached configuration data of a specified LPI, causing the GIC to reload the interrupt configuration from the appropriate LPI Configuration table.

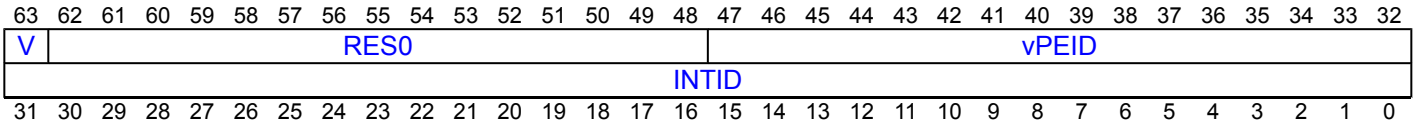
Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_INVLPIR is a 64-bit register.

Field descriptions



V, bit [63]

When GICv4.1 is implemented:

Indicates whether the INTID is virtual or physical.

V	Meaning
0b0	Invalidate is for a physical INTID.
0b1	Invalidate is for a virtual INTID.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

vPEID, bits [47:32]

When GICv4.1 is implemented:

When GICR_INVLPIR.V == 0, this field is RES0

When GICR_INVLPIR.V == 1, this field is the target vPEID of the invalidate.

Note

The size of this field is IMPLEMENTATION DEFINED, and is specified by the GICD_TYPER2.VIL and GICD_TYPER2.VID fields. Unimplemented bits are RES0.

Otherwise:

Reserved, RES0.

INTID, bits [31:0]

The INTID of the LPI to be invalidated.

Note

The size of this field is IMPLEMENTATION DEFINED, and is specified by the GICD_TYPER.IDbits field. Unimplemented bits are RES0.

Additional information

Note

If any LPI has been forwarded to the PE and a valid write to GICR_INVLPIR is received, the Redistributor must ensure it reloads its properties from memory and apply any changes by retrieving and reforwarding the LPI as required. This has no effect on the forwarded LPI if it has already been activated.

Accessing GICR_INVLPIR

When written with a 32-bit write the data is zero-extended to 64 bits.

This register is mandatory when any of the following are true:

- GICR_TYPER.Direct is 1.
- GICR_CTLR.IR is 1.
- GICv4.1 is implemented.

Otherwise, the functionality is IMPLEMENTATION DEFINED.

Writes to this register have no effect if either:

- The specified LPI is not currently stored in the local Redistributor.
- The INTID field corresponds to an unimplemented LPI.

GICR_INVLPIR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x00A0	GICR_INVLPIR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IPRIORITYR<n>, Interrupt Priority Registers, n = 0 - 7

The GICR_IPRIORITYR<n> characteristics are:

Purpose

Holds the priority of the corresponding interrupt for each SGI and PPI supported by the GIC.

Configuration

A copy of these registers is provided for each Redistributor.

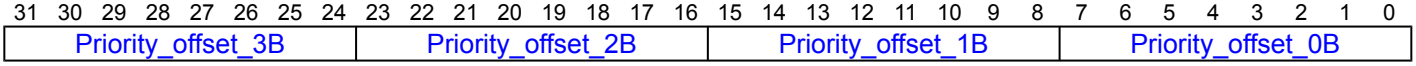
These registers are configured as follows:

- GICR_IPRIORITYR0-GICR_IPRIORITYR3 store the priority of SGIs.
- GICR_IPRIORITYR4-GICR_IPRIORITYR7 store the priority of PPIs.

Attributes

GICR_IPRIORITYR<n> is a 32-bit register.

Field descriptions



Priority_offset_3B, bits [31:24]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_2B, bits [23:16]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_1B, bits [15:8]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_0B, bits [7:0]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_IPRIORITYR<n>

These registers are used when affinity routing is enabled for the Security state of the interrupt. When affinity routing is not enabled the bits corresponding to the interrupt are RAZ/WI and GICD_IPRIORITYR<n> provides equivalent functionality.

These registers are used for SGIs and PPIs only. Equivalent functionality for SPIs is provided by GICD_IPRIORITYR<n>.

These registers are byte-accessible.

When GICD_CTLR.DS == 0:

- A field that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in 'Software accesses of interrupt priority' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICR_IPRIORITYR<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	`0x0400` + (4 * n)	GICR_IPRIORITYR<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IPRIORITYR<n>E, Interrupt Priority Registers (extended PPI range), n = 8 - 23

The GICR_IPRIORITYR<n>E characteristics are:

Purpose

Holds the priority of the corresponding interrupt for each extended PPI supported by the GIC.

Configuration

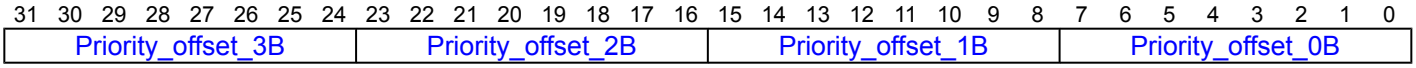
This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_IPRIORITYR<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_IPRIORITYR<n>E is a 32-bit register.

Field descriptions



Priority_offset_3B, bits [31:24]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_2B, bits [23:16]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_1B, bits [15:8]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_0B, bits [7:0]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt. For an INTID configured as non-maskable, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_IPRIORITYR<n> number, n, is given by n = (m-1024) DIV 4.
- The offset of the required GICR_IPRIORITYR<n>E register is (`0x400` + (4*n)).
- The byte offset of the required Priority field in this register is m MOD 4, where:
  - Byte offset 0 refers to register bits [7:0].
  - Byte offset 1 refers to register bits [15:8].
  - Byte offset 2 refers to register bits [23:16].
  - Byte offset 3 refers to register bits [31:24].

Accessing GICR_IPRIORITYR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISACTIVER<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0:

- A field that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in Software accesses of interrupt priority.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than once. The effect of the change must be visible in finite time.

GICR_IPRIORITYR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	`0x0400` + (4 * n)	GICR_IPRIORITYR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISACTIVER0, Interrupt Set-Active Register 0

The GICR_ISACTIVER0 characteristics are:

Purpose

Activates the corresponding SGI or PPI. These registers are used when saving and restoring GIC state.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISACTIVER0 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24
Set_active_bit31	Set_active_bit30	Set_active_bit29	Set_active_bit28	Set_active_bit27	Set_active_bit26	Set_active_bit25	Set_active_

Set_active_bit<x>, bit [x], for x = 31 to 0

Adds the active state to interrupt number x. Reads and writes have the following behavior:

Set_active_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is active, or is active and pending. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Accessing GICR_ISACTIVER0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISACTIVER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ISACTIVER<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_ISACTIVER<n>.

When GICD_CTLR.DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

The effect of a write must be visible in finite time. Reading back the written value from either the Set-Active or Clear-Active registers guarantees that a deactivate from a CPU interface Ordered-after the read, will observe the effects of the write on the Active state of the interrupt.

GICR_ISACTIVER0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0300	GICR_ISACTIVER0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISACTIVER<n>E, Interrupt Set-Active Registers, n = 1 - 2

The GICR_ISACTIVER<n>E characteristics are:

Purpose

Adds the active state to the corresponding PPI.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_ISACTIVER<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISACTIVER<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24
Set_active_bit31	Set_active_bit30	Set_active_bit29	Set_active_bit28	Set_active_bit27	Set_active_bit26	Set_active_bit25	Set_active_

Set_active_bit<x>, bit [x], for x = 31 to 0

For the extended PPIs, adds the active state to interrupt number x. Reads and writes have the following behavior:

Set_active_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is active, or active and pending on this PE. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ISACTIVER<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_ISACTIVER<n>E is (0x200 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.

Accessing GICR_ISACTIVER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISACTIVER<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

The effect of a write must be visible in finite time. Reading back the written value from either the Set-Active or Clear-Active registers guarantees that a deactivate from a CPU interface Ordered-after the read, will observe the effects of the write on the Active state of the interrupt.

GICR_ISACTIVER<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	`0x0300` + (4 * n)	GICR_ISACTIVER<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISENABLER0, Interrupt Set-Enable Register 0

The GICR_ISENABLER0 characteristics are:

Purpose

Enables forwarding of the corresponding SGI or PPI to the CPU interfaces.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISENABLER0 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	
Set_enable_bit31	Set_enable_bit30	Set_enable_bit29	Set_enable_bit28	Set_enable_bit27	Set_enable_bit26	Set_enable_bit25	Set_e

Set_enable_bit<x>, bit [x], for x = 31 to 0

For PPIs and SGIs, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

Set_enable_bit<x>	Meaning
0b0	If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.
0b1	If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Accessing GICR_ISENABLER0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISENABLER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ISENABLER<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_ISENABLER<n>.

When GICD_CTLR.DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ISENABLER0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0100	GICR_ISENABLER0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISENABLER<n>E, Interrupt Set-Enable Registers, n = 1 - 2

The GICR_ISENABLER<n>E characteristics are:

Purpose

Enables forwarding of the corresponding PPI to the CPU interfaces.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_ISENABLER<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISENABLER<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	
Set_enable_bit31	Set_enable_bit30	Set_enable_bit29	Set_enable_bit28	Set_enable_bit27	Set_enable_bit26	Set_enable_bit25	Set_e

Set_enable_bit<x>, bit [x], for x = 31 to 0

For the extended PPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

Set_enable_bit<x>	Meaning
0b0	If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.
0b1	If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ISENABLER<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_ISENABLER<n>E is (0x100 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.

Accessing GICR_ISENABLER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISENABLER<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ISENABLER<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	`0x0100` + (4 * n)	GICR_ISENABLER<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISPENDR0, Interrupt Set-Pending Register 0

The GICR_ISPENDR0 characteristics are:

Purpose

Adds the pending state to the corresponding SGI or PPI.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISPENDR0 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Set_pending_bit31	Set_pending_bit30	Set_pending_bit29	Set_pending_bit28	Set_pending_bit27	Set_pending_bit26	Set_pending_bi

Set_pending_bit<x>, bit [x], for x = 31 to 0

For PPIs and SGIs, adds the pending state to interrupt number x. Reads and writes have the following behavior:

Set_pending_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not pending on this PE. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is pending, or active and pending on this PE. If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases: • If the interrupt is already pending because of a write to GICR_ISPENDR0. • If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ISPENDR0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISPENDR0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ISPENDR<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_ISPENDR<n>.

When GICD_CTLR.DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ISPENDR0 can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance

GIC Redistributor	SGI_base	0x0200	GICR_ISPENDR0

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISPENDR<n>E, Interrupt Set-Pending Registers, n = 1 - 2

The GICR_ISPENDR<n>E characteristics are:

Purpose

Adds the pending state to the corresponding PPI.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_ISPENDR<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISPENDR<n>E is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25
Set_pending_bit31	Set_pending_bit30	Set_pending_bit29	Set_pending_bit28	Set_pending_bit27	Set_pending_bit26	Set_pending_bi

Set_pending_bit<x>, bit [x], for x = 31 to 0

For the extended PPIs, adds the pending state to interrupt number x. Reads and writes have the following behavior:

Set_pending_bit<x>	Meaning
0b0	If read, indicates that the corresponding interrupt is not pending on this PE. If written, has no effect.
0b1	If read, indicates that the corresponding interrupt is pending, or active and pending on this PE. If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases: • If the interrupt is already pending because of a write to GICR_ISPENDR<n>E. • If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ISPENDR<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_ISPENDR<n>E is (0x200 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.

Accessing GICR_ISPENDR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISPENDR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ISPENDR<n>E can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	`0x0200` + (4 * n)	GICR_ISPENDR<n>E

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_MPAMIDR, Report maximum PARTID and PMG Register

The GICR_MPAMIDR characteristics are:

Purpose

Reports the maximum support PARTID and PMG values.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_MPAMIDR are RES0.

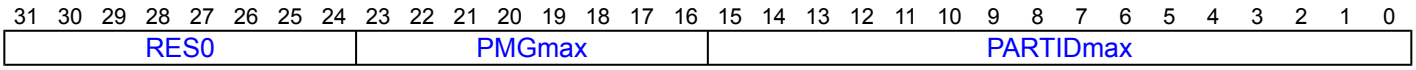
A copy of this register is provided for each Redistributor.

When GICR_TYPER.MPAM==0, this register is RES0.

Attributes

GICR_MPAMIDR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

PMGmax, bits [23:16]

Maximum PMG value supported.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PARTIDmax, bits [15:0]

Maximum PARTID value supported.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GICR_MPAMIDR

GICR_MPAMIDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x0018	GICR_MPAMIDR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_NSACR, Non-secure Access Control Register

The GICR_NSACR characteristics are:

Purpose

Enables Secure software to permit Non-secure software to create SGIs targeting the PE connected to this Redistributor by writing to ICC_SGI1R_EL1, ICC_ASGI1R_EL1 or ICC_SGI0R_EL1.

For more information, see 'Forwarding an SGI to a target PE' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Configuration

For a description on when a write to ICC_SGI0R_EL1, ICC_SGI1R_EL1 or ICC_ASGI1R_EL1 is permitted to generate an interrupt, see 'Use of control registers for SGI forwarding' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

GICR_NSACR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13
NS_access15		NS_access14		NS_access13		NS_access12		NS_access11		NS_access10		NS_access9		NS_access8		NS_access7		NS_ac

NS_access<x>, bits [2x+1:2x], for x = 15 to 0

Configures the level of Non-secure access permitted when the SGI is in Secure Group 0 or Secure Group 1, as defined from GICR_IGROUPR0 and GICR_IGRPMODR0. A field is provided for each SGI. The possible values of each 2-bit field are:

NS_access<x>	Meaning
0b00	Non-secure writes are not permitted to generate Secure Group 0 SGIs or Secure Group 1 SGIs.
0b01	Non-secure writes are permitted to generate a Secure Group 0 SGI.
0b10	As 0b01, but additionally Non-secure writes to are permitted to generate a Secure Group 1 SGI.
0b11	Reserved. If the field is programmed to the reserved value, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the valid values. However, to maintain the principle that as the value increases additional accesses are permitted Arm strongly recommends that implementations treat this value as 0b10. It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the valid value chosen.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_NSACR

This register is used when affinity routing is enabled. When affinity routing is not enabled for the Security state of the interrupt, GICD_NSACR<n> with n=0 provides equivalent functionality.

This register does not support PPIs.

GICR_NSACR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	SGI_base	0x0E00	GICR_NSACR

Accessible as follows:

- When GICD_CTLR.DS == 1, accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0 and an access is Secure, accesses to this register are **RW**.
- When GICD_CTLR.DS == 0 and an access is Non-secure, accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Root, accesses to this register are **RW**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Realm, accesses to this register are **RAZ/WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_PARTIDR, Set PARTID and PMG Register

The GICR_PARTIDR characteristics are:

Purpose

Sets the PARTID and PMG values used for memory accesses by the Redistributor.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GICR_PARTIDR are RES0.

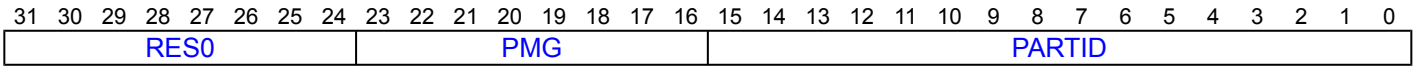
A copy of this register is provided for each Redistributor.

When GICR_TYPER.MPAM==0, this register is RES0.

Attributes

GICR_PARTIDR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

PMG value used when Redistributor accesses memory.

Bits not needed to represent PMG values in the range 0 to PMG_MAX are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'00000000'`.

PARTID, bits [15:0]

PARTID value used when Redistributor accesses memory.

Bits not needed to represent PARTID values in the range 0 to PARTID_MAX are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'0000000000000000'`.

Accessing GICR_PARTIDR

GICR_PARTIDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	`0x001C`	GICR_PARTIDR

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_PENDBASER, Redistributor LPI Pending Table Base Address Register

The GICR_PENDBASER characteristics are:

Purpose

Specifies the base address of the LPI Pending table, and the Shareability and Cacheability of accesses to the LPI Pending table.

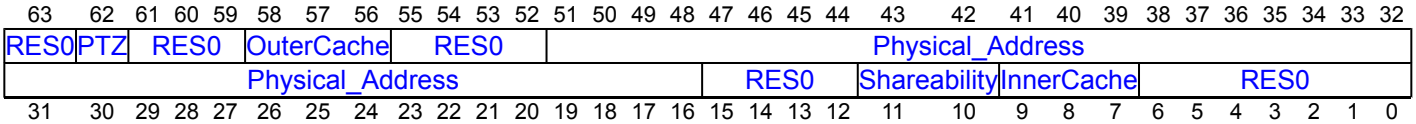
Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_PENDBASER is a 64-bit register.

Field descriptions



Bit [63]

Reserved, RES0.

PTZ, bit [62]

Pending Table Zero. Indicates to the Redistributor whether the LPI Pending table is zero when GICR_CTLR.EnableLPIs == 1.

This field is WO, and reads as 0.

PTZ	Meaning
0b0	The LPI Pending table is not zero, and contains live data.
0b1	The LPI Pending table is zero. Software must ensure the LPI Pending table is zero before this value is written.

Bits [61:59]

Reserved, RES0.

OuterCache, bits [58:56]

Indicates the Outer Cacheability attributes of accesses to the LPI Pending table.

OuterCache	Meaning
0b000	Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.
0b001	Normal Outer Non-cacheable.
0b010	Normal Outer Cacheable Read-allocate, Write-through.
0b011	Normal Outer Cacheable Read-allocate, Write-back.
0b100	Normal Outer Cacheable Write-allocate, Write-through.
0b101	Normal Outer Cacheable Write-allocate, Write-back.
0b110	Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52]

Reserved, RES0.

Physical_Address, bits [51:16]

Bits [51:16] of the physical address containing the LPI Pending table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [15:12]

Reserved, RES0.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the LPI Pending table.

Shareability	Meaning
0b00	Non-shareable.
0b01	Inner Shareable.
0b10	Outer Shareable.
0b11	Reserved. Treated as 0b00.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the LPI Pending table.

InnerCache	Meaning
0b000	Device-nGnRnE.
0b001	Normal Inner Non-cacheable.
0b010	Normal Inner Cacheable Read-allocate, Write-through.
0b011	Normal Inner Cacheable Read-allocate, Write-back.
0b100	Normal Inner Cacheable Write-allocate, Write-through.
0b101	Normal Inner Cacheable Write-allocate, Write-back.
0b110	Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [6:0]

Reserved, RES0.

Accessing GICR_PENDBASER

Having the GICR_PENDBASER OuterCache, Shareability or InnerCache fields programmed to different values on different Redistributors with GICR_CTLR.EnableLPIs == 1 in the system is UNPREDICTABLE.

Changing GICR_PENDBASER with GICR_CTLR.EnableLPIs == 1 is UNPREDICTABLE.

GICR_PENDBASER can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x0078	GICR_PENDBASER

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_PROPBASER, Redistributor Properties Base Address Register

The GICR_PROPBASER characteristics are:

Purpose

Specifies the base address of the LPI Configuration table, and the Shareability and Cacheability of accesses to the LPI Configuration table.

Configuration

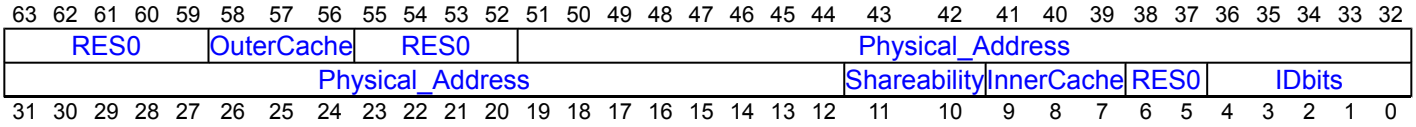
A copy of this register is provided for each Redistributor.

An implementation might make this register RO, for example to correspond to an LPI Configuration table in read-only memory.

Attributes

GICR_PROPBASER is a 64-bit register.

Field descriptions



Bits [63:59]

Reserved, RES0.

OuterCache, bits [58:56]

Indicates the Outer Cacheability attributes of accesses to the LPI Configuration table.

OuterCache	Meaning
0b000	Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.
0b001	Normal Outer Non-cacheable.
0b010	Normal Outer Cacheable Read-allocate, Write-through.
0b011	Normal Outer Cacheable Read-allocate, Write-back.
0b100	Normal Outer Cacheable Write-allocate, Write-through.
0b101	Normal Outer Cacheable Write-allocate, Write-back.
0b110	Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52]

Reserved, RES0.

Physical_Address, bits [51:12]

Bits [51:12] of the physical address containing the LPI Configuration table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the LPI Configuration table.

Shareability	Meaning
0b00	Non-shareable.
0b01	Inner Shareable.
0b10	Outer Shareable.
0b11	Reserved. Treated as 0b00.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the LPI Configuration table.

InnerCache	Meaning
0b000	Device-nGnRnE.
0b001	Normal Inner Non-cacheable.
0b010	Normal Inner Cacheable Read-allocate, Write-through.
0b011	Normal Inner Cacheable Read-allocate, Write-back.
0b100	Normal Inner Cacheable Write-allocate, Write-through.
0b101	Normal Inner Cacheable Write-allocate, Write-back.
0b110	Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IDbits, bits [4:0]

The number of bits of LPI INTID supported, minus one, by the LPI Configuration table starting at Physical_Address.

If the value of this field is larger than the value of GICD_TYPER.IDbits, the GICD_TYPER.IDbits value applies.

If the value of this field is less than 0b1101, indicating that the largest INTID is less than 8192 (the smallest LPI interrupt ID), the GIC will behave as if all physical LPIs are out of range.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_PROPBASER

It is IMPLEMENTATION DEFINED whether GICR_PROPBASER can be set to different values on different Redistributors. GICR_TYPER.CommonLPIAff identifies the Redistributors that must have GICR_PROPBASER set to the same values whenever GICR_CTLR.EnableLPIs == 1.

Setting different values in different copies of GICR_PROPBASER on Redistributors that are required to use a common LPI Configuration table when GICR_CTLR.EnableLPIs == 1 leads to UNPREDICTABLE behavior.

Other restrictions apply when a Redistributor caches information from GICR_PROPBASER. For more information, see 'LPI Configuration tables' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

GICR_PROPBASER can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x0070	GICR_PROPBASER

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_SETLPIR, Set LPI Pending Register

The GICR_SETLPIR characteristics are:

Purpose

Generates an LPI by setting the pending state of the specified LPI.

Configuration

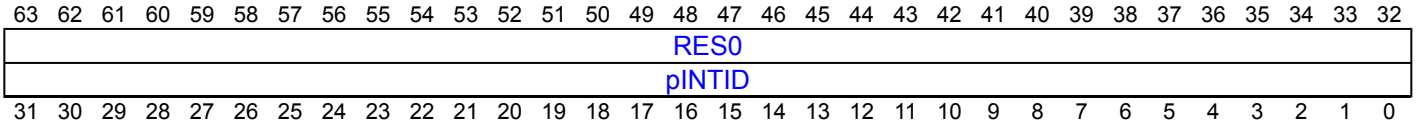
A copy of this register is provided for each Redistributor.

Attributes

GICR_SETLPIR is a 64-bit register.

Field descriptions

When GICR_TYPER.DirectLPI == 1:



Bits [63:32]

Reserved, RES0.

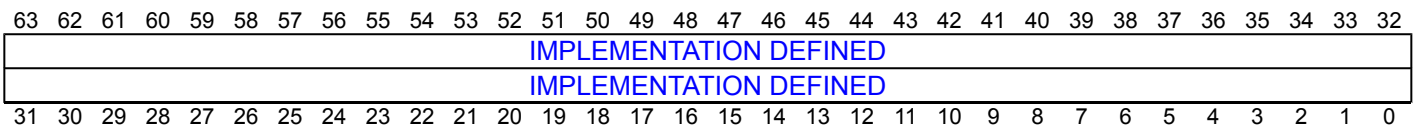
pINTID, bits [31:0]

The INTID of the physical LPI to be generated.

Note

The size of this field is IMPLEMENTATION DEFINED, and is specified by the GICD_TYPER.IDbits field. Unimplemented bits are RES0.

When GICR_TYPER.DirectLPI == 0:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing GICR_SETLPIR

When written with a 32-bit write the data is zero-extended to 64 bits.

This register is mandatory in an implementation that supports LPIs and does not include an ITS. The functionality is IMPLEMENTATION DEFINED in an implementation that does include an ITS.

Writes to this register have no effect if either:

- The pINTID field corresponds to an LPI that is already pending.
- The pINTID field corresponds to an unimplemented LPI.
- GICR_CTLR.EnableLPIs == 0.

GICR_SETLPIR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x0040	GICR_SETLPIR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_STATUSR, Error Reporting Status Register

The GICR_STATUSR characteristics are:

Purpose

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

Configuration

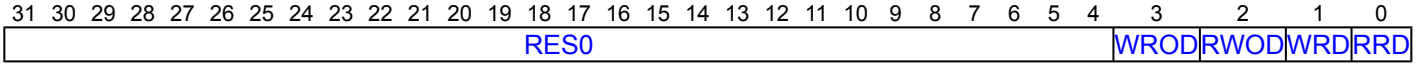
A copy of this register is provided for each Redistributor.

If the GIC implementation supports two Security states this register is Banked to provide Secure and Non-secure copies.

Attributes

GICR_STATUSR is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

WROD, bit [3]

Write to an RO location.

WROD	Meaning
0b0	Normal operation.
0b1	A write to an RO location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

RWOD, bit [2]

Read of a WO location.

RWOD	Meaning
0b0	Normal operation.
0b1	A read of a WO location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

WRD, bit [1]

Write to a reserved location.

WRD	Meaning
0b0	Normal operation.
0b1	A write to a reserved location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

RRD	Meaning
0b0	Normal operation.
0b1	A read of a reserved location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

Accessing GICR_STATUSR

This is an optional register. If the register is not implemented, the location is RAZ/WI.

GICR_STATUSR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x0010	GICR_STATUSR (S)

Accessible as follows:

- When an access is Secure, accesses to this register are **RW**.
- When FEAT_RME is implemented and an access is Root, accesses to this register are **RW**.

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x0010	GICR_STATUSR (NS)

Accessible as follows:

- When an access is Non-secure, accesses to this register are **RW**.
- When FEAT_RME is implemented and an access is Realm, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_SYNCR, Redistributor Synchronize Register

The GICR_SYNCR characteristics are:

Purpose

Indicates completion of register based invalidate operations.

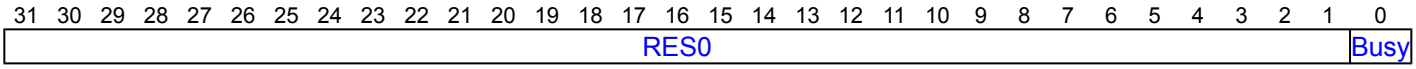
Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_SYNCR is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

Busy, bit [0]

Indicates completion of invalidation operations

Busy	Meaning
0b0	No operations are in progress.
0b1	A write is in progress to one or more of the following registers: • GICR_INVLPIR. • GICR_INVALLR. • GICv3, GICR_CLRLPIR.

This field tracks operations initiated on the same Redistributor.

Accessing GICR_SYNCR

When this register is accessed, it is optional that an implementation might wait until all operations are complete before returning a value, in which case GICR_SYNCR.Busy is always 0.

This register is mandatory when any of the following are true:

- GICR_TYPER.Direct is 1.
- GICR_CTLR.IR is 1.
- GICv4.1 is implemented.

Otherwise, the functionality is IMPLEMENTATION DEFINED.

GICR_SYNCR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x00C0	GICR_SYNCR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_TYPER, Redistributor Type Register

The GICR_TYPER characteristics are:

Purpose

Provides information about the configuration of this Redistributor.

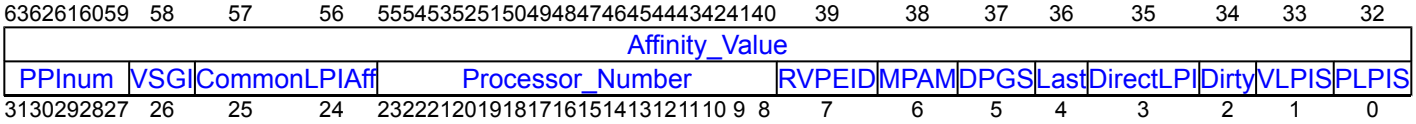
Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_TYPER is a 64-bit register.

Field descriptions



Affinity_Value, bits [63:32]

The identity of the PE associated with this Redistributor.

Bits [63:56] provide Aff3, the Affinity level 3 value for the Redistributor.

Bits [55:48] provide Aff2, the Affinity level 2 value for the Redistributor.

Bits [47:40] provide Aff1, the Affinity level 1 value for the Redistributor.

Bits [39:32] provide Aff0, the Affinity level 0 value for the Redistributor.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PPInum, bits [31:27]

When GICv3.1 is implemented:

The value derived from this field specifies the maximum PPI INTID that a GIC implementation can support. An implementation might not implement all PPIs up to this maximum.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PPInum	Meaning
0b00000	Maximum PPI INTID is 31.
0b00001	Maximum PPI INTID is 1087.
0b00010	Maximum PPI INTID is 1119.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

VSGI, bit [26]

When GICv4.1 is implemented:

Indicates whether vSGIs are supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VSGI	Meaning
0b0	Direct injection of SGIs not supported.
0b1	Direct injection of SGIs supported.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

CommonLPIAff, bits [25:24]

Indicates the scope of the CommonLPIAff group.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CommonLPIAff	Meaning
0b00	All Redistributors are members of the same CommonLPIAff group.
0b01	All Redistributors with the same Aff3 value are members of the same CommonLPIAff group.
0b10	All Redistributors with the same Aff3.Aff2 value are members of the same CommonLPIAff group.
0b11	All Redistributors with the same Aff3.Aff2.Aff1 value are members of the same CommonLPIAff group.

Redistributors in the same CommonLPIAff group must use the same copy of the LPI Configuration table, and if GICv4.1 is implemented the same copy of the vPE Configuration table.

Access to this field is **RO**.

Processor_Number, bits [23:8]

A unique identifier for the PE. When GITS_TYPER.PTA == 0, an ITS uses this field to identify the interrupt target.

When affinity routing is disabled for a Security state, this field indicates which GICD_ITARGETSR<n> corresponds to this Redistributor.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

RVPEID, bit [7]

When GICv4.1 is implemented:

Indicates how the resident vPE is specified.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RVPEID	Meaning
0b0	GICR_VPENDBASER records the address of the vPE's Virtual Pending Table.
0b1	GICR_VPENDBASER records vPEID.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

MPAM, bit [6]

When GICv3.1 is implemented:

MPAM

The value of this field is an IMPLEMENTATION DEFINED choice of:

MPAM	Meaning
0b0	MPAM not supported.
0b1	MPAM supported.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

DPGS, bit [5]

Sets support for GICR_CTLR.DPG* bits.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DPGS	Meaning
0b0	GICR_CTLR.DPG* bits are not supported.
0b1	GICR_CTLR.DPG* bits are supported.

Access to this field is **RO**.

Last, bit [4]

Indicates whether this Redistributor is the highest-numbered Redistributor in a series of contiguous Redistributor pages.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Last	Meaning
0b0	This Redistributor is not the highest-numbered Redistributor in a series of contiguous Redistributor pages.
0b1	This Redistributor is the highest-numbered Redistributor in a series of contiguous Redistributor pages.

Access to this field is **RO**.

DirectLPI, bit [3]

Indicates whether this Redistributor supports direct injection of LPIs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DirectLPI	Meaning
0b0	This Redistributor does not support direct injection of LPIs. The GICR_SETLPIR, GICR_CLRLPIR, GICR_INVLPIR, GICR_INVALLR, and GICR_SYNCR registers are either not implemented, or have an IMPLEMENTATION DEFINED purpose.
0b1	This Redistributor supports direct injection of LPIs. The GICR_SETLPIR, GICR_CLRLPIR, GICR_INVLPIR, GICR_INVALLR, and GICR_SYNCR registers are implemented.

Access to this field is **RO**.

Dirty, bit [2]

Controls the functionality of GICR_VPENDBASER.Dirty.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Dirty	Meaning
0b0	GICR_VPENDBASER.Dirty is UNKNOWN when GICR_VPENDBASER.Valid == 1.
0b1	GICR_VPENDBASER.Dirty indicates when the Virtual Pending Table has been parsed when GICR_VPENDBASER.Valid is written from 0 to 1.

When GICR_TYPER.VLPIS == 0, this field is RES0.

Note

In GICv4p1 implementations this field is RES1.

Access to this field is **RO**.

VLPIS, bit [1]

Indicates whether the GIC implementation supports virtual LPIs and the direct injection of virtual LPIs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VLPIS	Meaning
0b0	The implementation does not support virtual LPIs or the direct injection of virtual LPIs.
0b1	The implementation supports virtual LPIs and the direct injection of virtual LPIs.

Note

In GICv3 implementations this field is RES0.

Access to this field is **RO**.

PLPIS, bit [0]

Indicates whether the GIC implementation supports physical LPIs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PLPIS	Meaning
0b0	The implementation does not support physical LPIs.
0b1	The implementation supports physical LPIs.

Access to this field is **RO**.

Accessing GICR_TYPER

GICR_TYPER can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x0008	GICR_TYPER

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_VPENDBASER, Virtual Redistributor LPI Pending Table Base Address Register

The GICR_VPENDBASER characteristics are:

Purpose

Specifies the base address of the memory that holds the virtual LPI Pending table for the currently scheduled virtual machine.

Configuration

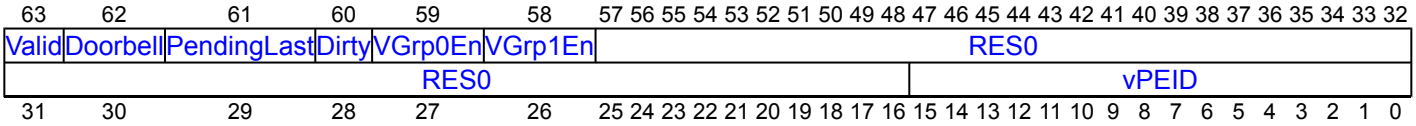
There are no configuration notes.

Attributes

GICR_VPENDBASER is a 64-bit register.

Field descriptions

When GICv4.1 is implemented:



Valid, bit [63]

This bit controls whether a vPE is scheduled:

Valid	Meaning
0b0	The virtual LPI Pending table is not valid. No vPE is scheduled.
0b1	The virtual LPI Pending table is valid. A vPE is scheduled.

Setting GICR_VPENDBASER.Valid == 1 when the associated CPU interface does not implement FEAT_GICv4 is UNPREDICTABLE.

Note

Software can determine whether a PE supports FEAT_GICv3 or FEAT_GICv4 by reading ID_AA64PFR0_EL1.

Writing a new value to any bit of GICR_VPENDBASER, other than GICR_VPENDBASER.Valid, when GICR_VPENDBASER.Valid==1 is UNPREDICTABLE.

Setting GICR_VPENDBASER.Valid to 1 is UNPREDICTABLE if GICR_VPROPBASER.Valid == 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Doorbell, bit [62]

When GICR_VPENDBASER.Valid is written from 1 to 0, this bit controls whether a default doorbell interrupt is requested for the descheduled vPE.

Doorbell	Meaning
0b0	No default doorbell requested.
0b1	Default doorbell requested.

When GICR_VPENDBASER.Valid is written from 1 to 0, if there are outstanding enabled pending interrupts then this bit is treated as 0.

When GICR_VPENDBASER.Valid is written from 1 to 0, if GICR_VPENDBASER.PendingLast is written as 1 then this bit is treated as 0.

When GICR_VPENDBASER.Valid == 1, reads return an UNKNOWN value.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

PendingLast, bit [61]

Indicates whether there are pending and enabled interrupts for the last scheduled vPE.

This value is set by the implementation when GICR_VPENDBASER.Valid is written from 1 to 0 and is otherwise UNKNOWN.

PendingLast	Meaning
0b0	There are no pending and enabled interrupts for the last scheduled vPE.
0b1	There is at least one pending and enabled interrupt for the last scheduled vPE.

When the GICR_VPENDBASER.Valid bit is written from 0 to 1, this bit is RES1.

When GICR_VPENDBASER.Valid is written from 1 to 0, if GICR_VPENDBASER.PendingLast is written as 1, then this bit is set to an UNKNOWN value.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Dirty, bit [60]

When GICR_VPENDBASER.Valid == 1:

Read-only. Reports whether the Virtual Pending table has been parsed.

Dirty	Meaning
0b0	Parsing of the Virtual Pending Table is complete.
0b1	Parsing of the Virtual Pending Table has not completed.

Writing 0 to GICR_VPENDBASER.Valid is UNPREDICTABLE while GICR_VPENDBASER.Dirty == 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Otherwise:

Read-only. Indicates whether a descheduling operation is in progress.

Dirty	Meaning
0b0	No descheduling operation in progress.
0b1	Descheduling operation in progress.

Writing 1 to GICR_VPENDBASER.Valid is UNPREDICTABLE while GICR_VPENDBASER.Dirty == 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

VGrp0En, bit [59]

Enable virtual Group 0 interrupts.

VGrp0En	Meaning
0b0	Forwarding of virtual Group 0 interrupts disabled.
0b1	Forwarding of virtual Group 0 interrupts enabled.

Writing a new value to VGrp0En while GICR_VPENDBASER.Valid==1 is CONSTRAINED UNPREDICTABLE:

- The update is ignored.
- The update is ignored for all purposes other than a direct read of the register.
- The virtual group enable is updated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

VGrp1En, bit [58]

Enable virtual Group 1 interrupts.

VGrp1En	Meaning
0b0	Forwarding of virtual Group 1 interrupts disabled.
0b1	Forwarding of virtual Group 1 interrupts enabled.

Writing a new value to VGrp1En while GICR_VPENDBASER.Valid==1 is CONSTRAINED UNPREDICTABLE:

- The update is ignored.
- The update is ignored for all purposes other than a direct read of the register.
- The virtual group enable is updated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Bits [57:16]

Reserved, RES0.

vPEID, bits [15:0]

When GICR_VPENDBASER.Valid == 1, ID of scheduled vPE.

When GICR_VPENDBASER.Valid == 1, if GICR_VPENDBASER.vPEID is set to a value greater than the configured vPEID width, the behavior of this field is CONSTRAINED UNPREDICTABLE:

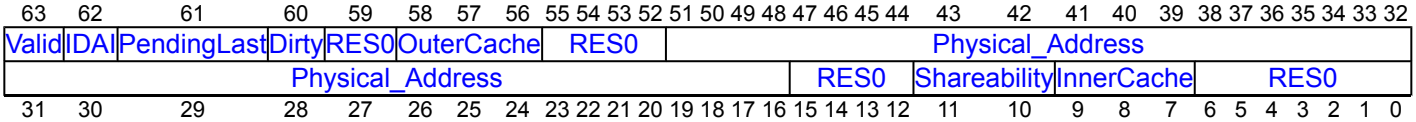
- GICR_VPENDBASER.vPEID is treated as having an UNKNOWN valid value for all purposes other than a direct read of the register.
- GICR_VPENDBASER.Valid is treated as being set to 0 for all purposes other than a direct read of the register.

Writing a new value to vPEID while GICR_VPENDBASER.Valid == 1 is CONSTRAINED UNPREDICTABLE:

- The update is ignored.
- The update is ignored for all purposes other than a direct read of the register.
- The new value is used.

The size of this field is IMPLEMENTATION DEFINED, and is specified by the GICD_TYPER2.VIL and GICD_TYPER2.VID fields, unimplemented bits are RES0.

When GICv4 is implemented:



Valid, bit [63]

This bit controls whether the virtual LPI Pending table is valid.

Valid	Meaning
0b0	The virtual LPI Pending table is not valid. No vPE is scheduled.
0b1	The virtual LPI Pending table is valid. A vPE is scheduled.

Setting GICR_VPENDBASER.Valid == 1 when the associated CPU interface does not implement FEAT_GICv4 is UNPREDICTABLE.

Note

Software can determine whether a PE supports FEAT_GICv3 or FEAT_GICv4 by reading ID_AA64PFR0_EL1.

Writing a new value to any bit of GICR_VPENDBASER, other than GICR_VPENDBASER.Valid, when GICR_VPENDBASER.Valid==1 is UNPREDICTABLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

IDAI, bit [62]

Implementation Defined Area Invalid. Indicates whether the IMPLEMENTATION DEFINED area in the virtual LPI Pending table is valid.

IDAI	Meaning
0b0	The IMPLEMENTATION DEFINED area is valid.
0b1	The IMPLEMENTATION DEFINED area is invalid and all pending interrupt information is held in the architecturally defined part of the virtual LPI Pending table.

For more information, see 'LPI Pending tables' and 'Virtual LPI Configuration tables and virtual LPI Pending tables' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

PendingLast, bit [61]

Indicates whether there are pending and enabled interrupts for the last scheduled vPE.

This value is set by the implementation when GICR_VPENDBASER.Valid has been written from 1 to 0 and is otherwise UNKNOWN.

PendingLast	Meaning
0b0	There are no pending and enabled interrupts for the last scheduled vPE.
0b1	There is at least one pending interrupt for the last scheduled vPE. It is IMPLEMENTATION DEFINED whether this bit is set when the only pending interrupts for the last scheduled vPE are not enabled. Arm deprecates setting PendingLast to 1 when the only pending interrupts for the last scheduled virtual machine are not enabled.

When the GICR_VPENDBASER.Valid bit is written from 0 to 1, this bit is RES1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Dirty, bit [60]

When GICR_VPENDBASER.Valid == 0:

Indicates whether a descheduling operation is in progress.

This field is read-only.

Dirty	Meaning
0b0	No descheduling operation in progress.
0b1	Descheduling operation in progress.

Writing 1 to GICR_VPENDBASER.Valid is UNPREDICTABLE while GICR_VPENDBASER.Dirty==1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

When GICR_VPENDBASER.Valid == 1 and GICR_TYPER.Dirty == 1:

This field is read-only. Reports whether the Virtual Pending table has been parsed.

Dirty	Meaning
0b0	Parsing of the Virtual Pending Table has completed.
0b1	Parsing of the Virtual Pending Table has not completed.

Writing 1 to GICR_VPENDBASER.Valid is UNPREDICTABLE while GICR_VPENDBASER.Dirty == 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Otherwise:

This field is read-only. This fields is UNKNOWN.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Bit [59]

Reserved, RES0.

OuterCache, bits [58:56]

Indicates the Outer Cacheability attributes of accesses to virtual LPI Pending tables of vPEs targeting this Redistributor.

OuterCache	Meaning
0b000	Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.
0b001	Normal Outer Non-cacheable.
0b010	Normal Outer Cacheable Read-allocate, Write-through.
0b011	Normal Outer Cacheable Read-allocate, Write-back.
0b100	Normal Outer Cacheable Write-allocate, Write-through.
0b101	Normal Outer Cacheable Write-allocate, Write-back.
0b110	Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The Cacheability, Outer Cacheability and Shareability fields are used for accesses to the virtual LPI Pending table of resident and non-resident vPEs.

If the OuterCacheabilty attribute of the virtual LPI Pending tables that are associated with vPEs targeting the same Redistributor are different, behavior is UNPREDICTABLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52]

Reserved, RES0.

Physical_Address, bits [51:16]

Bits [51:16] of the physical address containing the virtual LPI Pending table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [15:12]

Reserved, RES0.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the virtual LPI Pending table.

Shareability	Meaning
0b00	Non-shareable.
0b01	Inner Shareable.
0b10	Outer Shareable.
0b11	Reserved. Treated as 0b00.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The Cacheability, Outer Cacheability and Shareability fields are used for accesses to the virtual LPI Pending table of resident and non-resident vPEs.

If the Shareability attribute of the virtual LPI Pending tables that are associated with vPEs targeting the same Redistributor are different, behavior is UNPREDICTABLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the virtual LPI Pending table.

InnerCache	Meaning
0b000	Device-nGnRnE.
0b001	Normal Inner Non-cacheable.
0b010	Normal Inner Cacheable Read-allocate, Write-through.
0b011	Normal Inner Cacheable Read-allocate, Write-back.
0b100	Normal Inner Cacheable Write-allocate, Write-through.
0b101	Normal Inner Cacheable Write-allocate, Write-back.
0b110	Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.

The Cacheability, Outer Cacheability and Shareability fields are used for accesses to the virtual LPI Pending table of resident and non-resident vPEs.

If the InnerCacheabilty attribute of the virtual LPI Pending tables that are associated with vPEs targeting the same Redistributor are different, behavior is UNPREDICTABLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [6:0]

Reserved, RES0.

Accessing GICR_VPENDBASER

The effect of a write to this register is not guaranteed to be visible throughout the affinity hierarchy, as indicated by GICR_CTLR.RWP == 0.

GICR_VPENDBASER can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	VLPI_base	0x0078	GICR_VPENDBASER

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_VPROPBASER, Virtual Redistributor Properties Base Address Register

The GICR_VPROPBASER characteristics are:

Purpose

In GICv4.0, specifies the base address of the memory that holds the virtual LPI Configuration table for the currently scheduled virtual machine.

In GICv4.1, specifies the base address of the memory that holds the vPE Configuration table.

Configuration

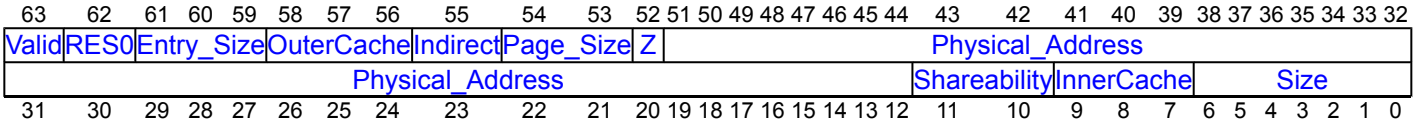
This register is provided in FEAT_GICv4 implementations only.

Attributes

GICR_VPROPBASER is a 64-bit register.

Field descriptions

When GICv4.1 is implemented:



Valid, bit [63]

This bit controls whether the vPE Configuration Table is valid.

Valid	Meaning
0b0	The vPE Configuration table is not valid.
0b1	The vPE Configuration table is valid.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Bit [62]

Reserved, RES0.

Entry_Size, bits [61:59]

Specifies the number 64-bit doublewords per table entry, minus one.

This bit is read-only.

OuterCache, bits [58:56]

Indicates the Outer Cacheability attributes of accesses to the table.

OuterCache	Meaning
0b000	Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.
0b001	Normal Outer Non-cacheable.
0b010	Normal Outer Cacheable Read-allocate, Write-through.
0b011	Normal Outer Cacheable Read-allocate, Write-back.
0b100	Normal Outer Cacheable Write-allocate, Write-through.
0b101	Normal Outer Cacheable Write-allocate, Write-back.
0b110	Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Indirect, bit [55]

This field indicates whether GICR_VPROPBASER specifies a single, flat table or a two-level table where the first level contains a list of descriptors.

Indirect	Meaning
0b0	Single Level. The Size field indicates the number of pages used to store data associated with each table entry.
0b1	Two Level. The Size field indicates the number of pages that contain an array of 64-bit descriptors to pages that are used to store the data associated with each table entry. A little endian memory order model is used.

This field is RAZ/WI for GIC implementations that only support flat tables.

If the supported vPEID width indicated by GICD_TYPER2.VIL and GICD_TYPER2.VID, and the smallest page size that is supported result in a single level table that requires multiple pages, then implementing this bit as RAZ/WI is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Page_Size, bits [54:53]

The following values indicate the size of page that the translation table uses:

Page_Size	Meaning
0b00	4KB.
0b01	16KB.
0b10	64KB.
0b11	Reserved. Treated as 0b10.

Note

If the GIC implementation supports only a single, fixed page size, this field might be RO.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Z, bit [52]

When GICR_VPROPBASER.Valid is written from 0 to 1, GICR_VPROPBASER.Z indicates whether the vPE Configuration table is known to contain all zeros.

Z	Meaning
0b0	The vPE Configuration table is not zero, and contains live data.
0b1	The vPE Configuration table is zero.

Setting GICR_VPROPBASER.Z to 0 causes the IRI to reload configuration from memory

When GICR_VPROPBASER.Valid is written from 0 to 1, if GICR_VPROPBASER.Z==1 behavior is UNPREDICTABLE if the allocated memory does not contain all zeros.

This field is WO, and reads as 0.

Physical_Address, bits [51:12]

Bits [51:12] of the physical address containing the vPE Configuration table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the vPE Configuration table.

Shareability	Meaning
0b00	Non-shareable.
0b01	Inner Shareable.
0b10	Outer Shareable.
0b11	Reserved. Treated as 0b00.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the vPE Configuration table.

InnerCache	Meaning
0b000	Device-nGnRnE.
0b001	Normal Inner Non-cacheable.
0b010	Normal Inner Cacheable Read-allocate, Write-through.
0b011	Normal Inner Cacheable Read-allocate, Write-back.
0b100	Normal Inner Cacheable Write-allocate, Write-through.
0b101	Normal Inner Cacheable Write-allocate, Write-back.
0b110	Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Size, bits [6:0]

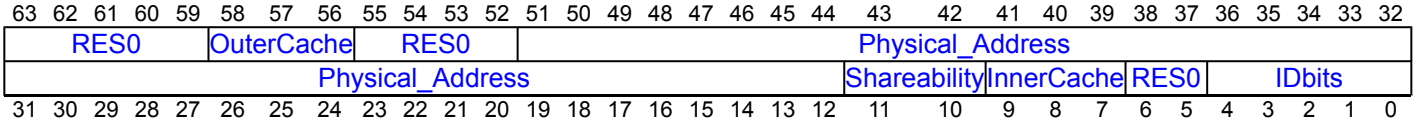
The number of pages of physical memory allocated to the table, minus one.

GICR_VPROPBASER.Page_Size specifies the size of each page.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

When GICv4 is implemented:



Bits [63:59]

Reserved, RES0.

OuterCache, bits [58:56]

Indicates the Outer Cacheability attributes of accesses to the LPI Configuration table.

OuterCache	Meaning
0b000	Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.
0b001	Normal Outer Non-cacheable.
0b010	Normal Outer Cacheable Read-allocate, Write-through.
0b011	Normal Outer Cacheable Read-allocate, Write-back.
0b100	Normal Outer Cacheable Write-allocate, Write-through.
0b101	Normal Outer Cacheable Write-allocate, Write-back.
0b110	Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52]

Reserved, RES0.

Physical_Address, bits [51:12]

Bits [51:12] of the physical address containing the virtual LPI Configuration table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the vPE Configuration table.

Shareability	Meaning
0b00	Non-shareable.
0b01	Inner Shareable.
0b10	Outer Shareable.
0b11	Reserved. Treated as 0b00.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the vPE Configuration table.

InnerCache	Meaning
0b000	Device-nGnRnE.
0b001	Normal Inner Non-cacheable.
0b010	Normal Inner Cacheable Read-allocate, Write-through.
0b011	Normal Inner Cacheable Read-allocate, Write-back.
0b100	Normal Inner Cacheable Write-allocate, Write-through.
0b101	Normal Inner Cacheable Write-allocate, Write-back.
0b110	Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IDbits, bits [4:0]

The number of bits of virtual LPI INTID supported, minus one.

If the value of this field is less than 0b1101, indicating that the largest INTID is less than 8192 (the smallest LPI interrupt ID), the GIC will behave as if all virtual LPIs are out of range.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_VPROPBASER

GICR_VPROPBASER can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	VLPI_base	0x0070	GICR_VPROPBASER

Accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_VSGIPENDR, Redistributor virtual SGI pending state register

The GICR_VSGIPENDR characteristics are:

Purpose

Requests the pending state of virtual SGIs for a specified vPE.

Configuration

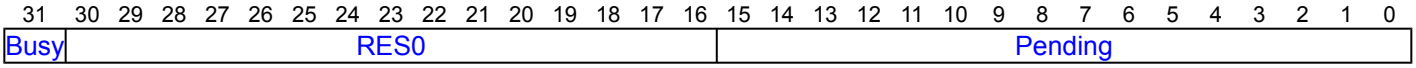
This register is present only when GICv4.1 is implemented. Otherwise, direct accesses to GICR_VSGIPENDR are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_VSGIPENDR is a 32-bit register.

Field descriptions



Busy, bit [31]

ID of target vPEID

Busy	Meaning
0b0	Query of virtual SGI state not in progress.
0b1	Query of virtual SGI state in progress.

Bits [30:16]

Reserved, RES0.

Pending, bits [15:0]

Pending state of virtual SGIs for requested vPEID.

This field is UNKNOWN when GICR_VSGIPENDR.Busy == 1

Accessing GICR_VSGIPENDR

GICR_VSGIPENDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	VLPI_base	0x0088	GICR_VSGIPENDR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_VSGIR, Redistributor virtual SGI pending state request register

The GICR_VSGIR characteristics are:

Purpose

Requests the pending state of virtual SGIs for a specified vPE.

Configuration

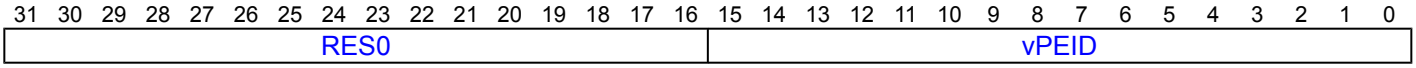
This register is present only when GICv4.1 is implemented. Otherwise, direct accesses to GICR_VSGIR are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_VSGIR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

vPEID, bits [15:0]

ID of target vPE

Writing this field is CONSTRAINED UNPREDICTABLE when GICR_VSGIPENDR.Busy == 1, with either the write ignored or a new query started.

Writing a value greater than the configured vPEID width behavior is CONSTRAINED UNPREDICTABLE, with either:

- vPEID is treated as having an UNKNOWN valid value.
- The write is ignored.

The size of this field is IMPLEMENTATION DEFINED, and is specified by the GICD_TYPER2.VIL and GICD_TYPER2.VID fields. Unimplemented bits are RES0.

Accessing GICR_VSGIR

GICR_VSGIR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	VLPI_base	0x0080	GICR_VSGIR

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_WAKER, Redistributor Wake Register

The GICR_WAKER characteristics are:

Purpose

Permits software to control the behavior of the WakeRequest power management signal corresponding to the Redistributor. Power management operations follow the rules in 'Power management' in in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_WAKER is a 32-bit register.

Field descriptions

31	30:3	2	1	0
IMPLEMENTATION DEFINED	RES0	ChildrenAsleep	ProcessorSleep	IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED, bit [31]

IMPLEMENTATION DEFINED.

Bits [30:3]

Reserved, RES0.

ChildrenAsleep, bit [2]

Read-only. Indicates whether the connected PE is quiescent:

ChildrenAsleep	Meaning
0b0	An interface to the connected PE might be active.
0b1	All interfaces to the connected PE are quiescent.

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

ProcessorSleep, bit [1]

Indicates whether the Redistributor can assert the **WakeRequest** signal:

ProcessorSleep	Meaning
0b0	This PE is not in, and is not entering, a low power state.
0b1	The PE is either in, or is in the process of entering, a low power state. All interrupts that arrive at the Redistributor: • Assert a **WakeRequest** signal. • Are held in the pending state at the Redistributor, and are not communicated to the CPU interface. **Note** When ProcessorSleep == 1, the Redistributor must ensure that any interrupts that are pending on the CPU interface are released. For an implementation that is using the GIC Stream Protocol Interface: • A Quiesce command puts the interface between the Redistributor and the CPU interface in a quiescent state. For more information, see 'Quiesce (IRI)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069). • A Release command releases any interrupts that are pending on the CPU interface. For more information, see 'Release (ICC)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Before powering down a PE, software must set this bit to 1 and wait until ChildrenAsleep == 1. After powering up a PE, or following a failed powerdown, software must set this bit to 0 and wait until ChildrenAsleep == 0.

Changing ProcessorSleep from 1 to 0 when ChildrenAsleep is not 1 results in UNPREDICTABLE behavior.

Changing ProcessorSleep from 0 to 1 when the Enable for each interrupt group in the associated CPU interface is not 0 results in UNPREDICTABLE behavior.

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

IMPLEMENTATION DEFINED, bit [0]

IMPLEMENTATION DEFINED.

Accessing GICR_WAKER

To ensure a Redistributor is quiescent, software must write to GICR_WAKER with ProcessorSleep == 1, then poll the register until ChildrenAsleep == 1.

Resetting the connected PE when GICR_WAKER.ProcessorSleep==0 or GICR_WAKER.ChildrenAsleep==0, can lead to UNPREDICTABLE behavior in the IRI.

Resetting the IRI when GICR_WAKER.ProcessorSleep==0 or GICR_WAKER.ChildrenAsleep==0 can lead to UNPREDICTABLE behavior in the connected PE.

GICR_WAKER can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
GIC Redistributor	RD_base	0x0014	GICR_WAKER

Accessible as follows:

- When GICD_CTLR.DS == 1, accesses to this register are **RW**.
- When GICD_CTLR.DS == 0 and an access is Secure, accesses to this register are **RW**.
- When GICD_CTLR.DS == 0 and an access is Non-secure, accesses to this register are **RAZ/WI**.

- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Root, accesses to this register are **RW**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented, and an access is Realm, accesses to this register are **RAZ/WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_ABPR, Virtual Machine Aliased Binary Point Register

The GICV_ABPR characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

This register corresponds to GICC_ABPR in the physical CPU interface.

Note

GICH_LR<n>.Group determines whether a virtual interrupt is Group 0 or Group 1.

Configuration

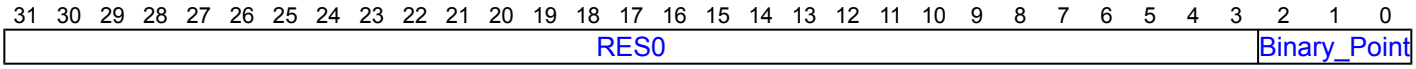
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_ABPR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_ABPR is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

Binary_Point, bits [2:0]

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

For information about how this field determines the interrupt priority bits assigned to the group priority field, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to `'000'`.

Additional information

The Binary_Point field of this register is aliased to GICH_VMCR.VBPR1.

Accessing GICV_ABPR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_BPR1 provides equivalent functionality.

- For AArch64 implementations, ICC_BPR1_EL1 provides equivalent functionality.

The value contained in this register is one greater than the actual applied binary point value, as described in 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This register is used for Group 1 interrupts when GICV_CTLR.CBPR == 0. GICV_BPR provides equivalent functionality for Group 0 interrupts, and for Group 1 interrupts when GICV_CTLR.CBPR == 1.

GICV_ABPR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x001C	GICV_ABPR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_AEOIR, Virtual Machine Aliased End Of Interrupt Register

The GICV_AEOIR characteristics are:

Purpose

A write to this register performs a priority drop for the specified Group 1 virtual interrupt and, if GICV_CTLR.EOImode == 0, also deactivates the interrupt.

Configuration

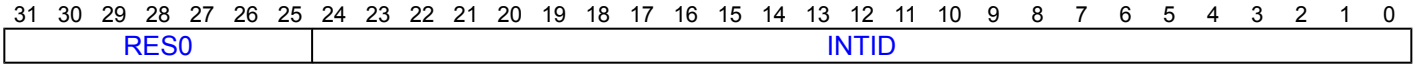
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_AEOIR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_AEOIR is a 32-bit register.

Field descriptions



Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Additional information

A successful EOI request means that:

- The highest priority bit in GICH_APR<n> is cleared, causing the running priority to drop.
- If the appropriate GICV_CTLR.EOImode bit == 0, the interrupt is deactivated in the corresponding List register. If the INTID corresponds to a hardware interrupt, the interrupt is also deactivated in the Distributor.

Note

Only Group 1 interrupts can target the hypervisor, and therefore only Group 1 interrupts are deactivated in the Distributor.

A write to this register is UNPREDICTABLE if the INTID corresponds to a Group 0 interrupt. In addition, the following GICv2 UNPREDICTABLE cases require specific actions:

- If highest active priority is Group 0 and the identified interrupt is in the List Registers and it matches the highest active priority. When EL2 is using System registers and ICH_VTR_EL2.SEIS is 1, an IMPLEMENTATION DEFINED SEI might be generated, otherwise GICv3 implementations must ignore such writes.
- If the identified interrupt is in the List Registers, and the HW bit is 1, and the interrupt to be deactivated is an SGI (that is, the value of Physical_ID is between 0 and 15). GICv3 implementations must perform the deactivate operation. This means that a GICv3 implementation in legacy operation must ensure only a single SGI is active for a PE.
- If the identified interrupt is in the List Registers, and the HW bit is 1, and the corresponding pINTID field value is between 1020 and 1023, indicating a special purpose INTID. GICv3 implementations must not perform a deactivate operation but must still change the state of the List register as appropriate. When EL2 is using System registers and ICH_VTR_EL2.SEIS is 1, an implementation might generate a system error.

Accessing GICV_AEOIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_EOIR1 provides equivalent functionality.
- For AArch64 implementations, ICC_EOIR1_EL1 provides equivalent functionality.

This register is used for Group 1 interrupts only. GICV_EOIR provides equivalent functionality for Group 0 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_AEOIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x0024	GICV_AEOIR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **WO**.
- When an access is Secure, accesses to this register are **WO**.
- When an access is Non-secure, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_AHPPIR, Virtual Machine Aliased Highest Priority Pending Interrupt Register

The GICV_AHPPIR characteristics are:

Purpose

Provides the INTID of the highest priority pending Group 1 virtual interrupt in the List registers.

This register corresponds to the physical CPU interface register GICC_AHPPIR.

Configuration

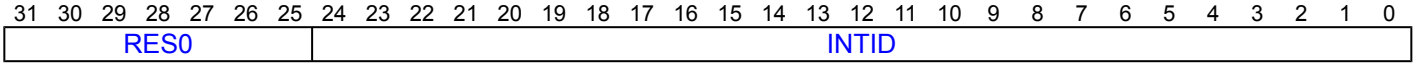
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_AHPPIR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_AHPPIR is a 32-bit register.

Field descriptions



Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Additional information

A read of this register returns the spurious INTID 1023 if any of the following are true:

- There are no pending interrupts of sufficiently high priority value to be signaled to the PE.
- The highest priority pending interrupt is in Group 0.

Accessing GICV_AHPPIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_HPPIR1 provides equivalent functionality.
- For AArch64 implementations, ICC_HPPIR1_EL1 provides equivalent functionality.

This register is used for Group 1 interrupts only. GICV_HPPIR provides equivalent functionality for Group 0 interrupts.

The register does not return the INTID of an interrupt that is active and pending.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_AHPPIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x0028	GICV_AHPPIR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_AIAR, Virtual Machine Aliased Interrupt Acknowledge Register

The GICV_AIAR characteristics are:

Purpose

Provides the INTID of the signaled Group 1 virtual interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

This register corresponds to the physical CPU interface register GICC_AIAR.

Configuration

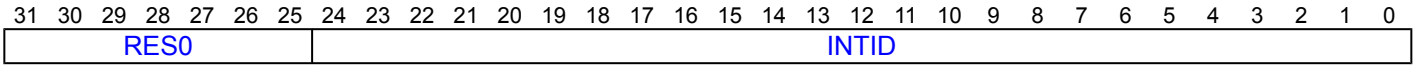
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_AIAR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_AIAR is a 32-bit register.

Field descriptions



Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Additional information

The operation of this register is similar to the operation of GICV_IAR. When a vPE reads this register, the corresponding GICH_LR<n>.Group field is checked to determine whether the interrupt is in Group 0 or Group 1:

- If the interrupt is Group 0, the spurious INTID 1023 is returned and the interrupt is not acknowledged.
- If the interrupt is Group 1, the INTID is returned. The List register entry is updated to active state, and the appropriate bit in GICH_APR<n> is set to 1.

A read of this register returns the spurious INTID 1023 if any of the following are true:

- When the virtual CPU interface is enabled and GICH_HCR.En == 1:

    - There are no pending interrupts of sufficiently high priority value to be signaled to the PE.
    - The highest priority pending interrupt is in Group 0.
- Interrupt signaling by the virtual CPU interface is disabled.

Accessing GICV_AIAR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_IAR1 provides equivalent functionality.
- For AArch64 implementations, ICC_IAR1_EL1 provides equivalent functionality.

This register is used for Group 1 interrupts only. GICV_IAR provides equivalent functionality for Group 0 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_AIAR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x0020	GICV_AIAR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_APR<n>, Virtual Machine Active Priorities Registers, n = 0 - 3

The GICV_APR<n> characteristics are:

Purpose

Provides information about interrupt active priorities.

These registers correspond to the physical CPU interface registers GICC_APR<n>.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_APR<n> are RES0.

When System register access is disabled for EL2, these registers access GICH_APR<n>, and all active priorities for virtual machines are held in GICH_APR<n> regardless of interrupt group.

When System register access is enabled for EL2, these registers access ICH_AP1R<n>_EL2, and all active priorities for virtual machines are held in ICH_AP1R<n>_EL2 regardless of interrupt group.

Attributes

GICV_APR<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<x>, bit [x], for x = 31 to 0

Provides information about active priorities for the virtual machine.

See GICH_APR<n> and ICH_AP1R<n>_EL2 for the correspondence between priorities and bits.

Accessing GICV_APR<n>

If System register access is not enabled for EL2, these registers access GICH_APR<n>. If System register access is enabled for EL2, these registers access ICH_AP1R<n>_EL2. All active priority mapped guests are held in the accessed registers, regardless of interrupt group.

GICV_APR<n> can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x00D0 + (4 * n)	GICV_APR<n>

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_BPR, Virtual Machine Binary Point Register

The GICV_BPR characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

This register corresponds to GICC_BPR in the physical CPU interface.

Note

GICH_LR<n>.Group determines whether a virtual interrupt is Group 0 or Group 1.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_BPR are RES0.

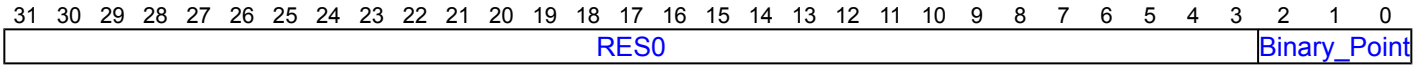
This register is available when the GIC implementation supports interrupt virtualization.

When GICV_CTLR.CBPR == 1, this register determines interrupt preemption for both Group 0 and Group 1 interrupts.

Attributes

GICV_BPR is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

Binary_Point, bits [2:0]

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

For information about how this field determines the interrupt priority bits assigned to the group priority field, see 'ICC_BPR0_EL1 Binary Point for Group 1 interrupts when CBPR == 1, or for Group 0 interrupts' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Additional information

The Binary_Point field of this register is aliased to GICH_VMCR.VBPR0.

Accessing GICV_BPR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_BPR0 provides equivalent functionality.
- For AArch64 implementations, ICC_BPR0_EL1 provides equivalent functionality.

GICV_BPR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x0008	GICV_BPR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_CTLR, Virtual Machine Control Register

The GICV_CTLR characteristics are:

Purpose

Controls the behavior of virtual interrupts.

This register corresponds to the physical CPU interface register GICC_CTLR.

Configuration

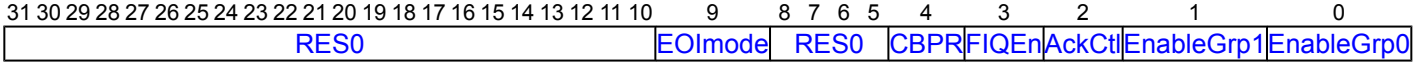
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_CTLR are RES0.

This register is available when a GIC implementation supports interrupt virtualization.

Attributes

GICV_CTLR is a 32-bit register.

Field descriptions



Bits [31:10]

Reserved, RES0.

EOImode, bit [9]

Controls the behavior associated with the GICV_EOIR, GICV_AEOIR, and GICV_DIR registers:

EOImode	Meaning
0b0	Writes to GICV_EOIR and GICV_AEOIR perform priority drop and deactivate interrupt operations simultaneously. Behavior on a write to GICV_DIR is unpredictable. When it has completed processing the interrupt, the virtual machine writes to GICV_EOIR or GICV_AEOIR to deactivate the interrupt. The write updates the List registers and causes the virtual CPU interface to signal the interrupt completion to the physical Distributor.
0b1	Writes to GICV_EOIR and GICV_AEOIR perform priority drop operation only. Writes to GICV_DIR perform deactivate interrupt operation only. When it has completed processing the interrupt, the virtual machine writes to GICV_DIR to deactivate the interrupt. The write updates the List registers and causes the virtual CPU interface to signal the interrupt completion to the Distributor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:5]

Reserved, RES0.

CBPR, bit [4]

Controls whether GICV_BPR affects both Group 0 and Group 1 interrupts:

CBPR	Meaning
0b0	GICV_BPR affects Group 0 virtual interrupts only. GICV_ABPR affects Group 1 virtual interrupts only.
0b1	GICV_BPR affects both Group 0 and Group 1 virtual interrupts.

For more information, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FIQEn, bit [3]

FIQ Enable. Controls whether Group 0 virtual interrupts are presented as virtual FIQs:

FIQEn	Meaning
0b0	Group 0 virtual interrupts are presented as virtual IRQs.
0b1	Group 0 virtual interrupts are presented as virtual FIQs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AckCtl, bit [2]

Arm deprecates use of this bit. Arm strongly recommends that software is written to operate with this bit always cleared to 0.

Acknowledge control. When the highest priority interrupt is Group 1, determines whether GICV_IAR causes the CPU interface to acknowledge the interrupt or returns the spurious identifier 1022, and whether GICV_HPPIR returns the interrupt ID or the special identifier 1022.

AckCtl	Meaning
0b0	If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an interrupt ID of 1022.
0b1	If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the interrupt ID of the corresponding interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EnableGrp1, bit [1]

Enables the signaling of Group 1 virtual interrupts by the virtual CPU interface to the virtual machine:

EnableGrp1	Meaning
0b0	Signaling of Group 1 interrupts is disabled.
0b1	Signaling of Group 1 interrupts is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EnableGrp0, bit [0]

Enables the signaling of Group 0 virtual interrupts by the virtual CPU interface to the virtual machine:

EnableGrp0	Meaning
0b0	Signaling of Group 0 interrupts is disabled.
0b1	Signaling of Group 0 interrupts is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICV_CTLR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_CTLR provides equivalent functionality.
- For AArch64 implementations, ICC_CTLR_EL1 provides equivalent functionality.

GICV_CTLR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x0000	GICV_CTLR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_DIR, Virtual Machine Deactivate Interrupt Register

The GICV_DIR characteristics are:

Purpose

Deactivates a specified virtual interrupt in the GICH_LR<n> List registers.

This register corresponds to the physical CPU interface register GICC_DIR.

Configuration

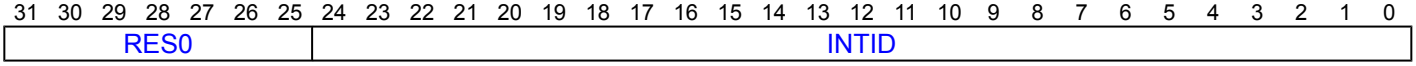
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_DIR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_DIR is a 32-bit register.

Field descriptions



Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Additional information

When the virtual machine writes to this register, the specified interrupt in the List registers is changed from active to inactive, or from active and pending to pending. If the specified interrupt is present in the List registers but is not in either the active or active and pending states, the effect is UNPREDICTABLE. If the specified interrupt is not present in the List registers, GICH_HCR.EOICount is incremented, potentially generating a maintenance interrupt.

Note

If the specified interrupt is not present in the List registers, the virtual machine cannot recover the INTID. Therefore, the hypervisor must ensure that, when GICV_CTLR.EOImode == 1, no more than one active interrupt is transferred from the List registers into a software list. If more than one

active interrupt that is not stored in the List registers exists, the hypervisor must handle accesses to GICV_DIR in software, typically by trapping these accesses.

If the corresponding GICH_LR<n>.HW == 1, indicating a hardware interrupt, then a deactivate request is sent to the physical Distributor, identifying the physical INTID from the corresponding field in the List register. This effect is identical to a Non-secure write to GICC_DIR from the PE having that physical INTID. This means that if the corresponding physical interrupt is marked as Group 0, the request is ignored.

Note

Interrupt deactivation using this register is based on the provided INTID, with no requirement to deactivate interrupts in any particular order. A single register is therefore used to deactivate both Group 0 and Group 1 interrupts.

Accessing GICV_DIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_DIR provides equivalent functionality.
- For AArch64 implementations, ICC_DIR_EL1 provides equivalent functionality.

Writes to this register are valid only when GICV_CTLR.EOImode == 1. Writes to this register are otherwise UNPREDICTABLE.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_DIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x1000	GICV_DIR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **WO**.
- When an access is Secure, accesses to this register are **WO**.
- When an access is Non-secure, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_EOIR, Virtual Machine End Of Interrupt Register

The GICV_EOIR characteristics are:

Purpose

A write to this register performs a priority drop for the specified Group 0 virtual interrupt and, if GICV_CTLR.EOImode == 0, also deactivates the interrupt.

This register corresponds to the physical CPU interface register GICC_EOIR.

Configuration

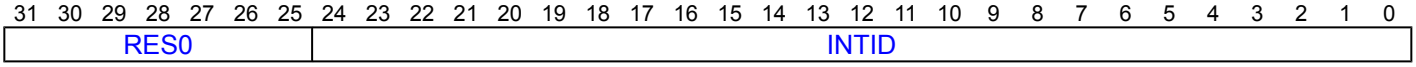
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_EOIR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_EOIR is a 32-bit register.

Field descriptions



Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Additional information

The behavior of this register depends on the setting of GICV_CTLR.EOImode:

GICV_CTLR.EOImode	Behavior
0b0	Both the priority drop and the deactivate interrupt effects occur
0b1	Only the priority drop effect occurs.

A successful EOI request means that:

- The highest priority bit in GICH_APR<n> is cleared, causing the running priority to drop.
- If the appropriate GICV_CTLR.EOImode bit == 0, the interrupt is deactivated in the corresponding List register GICH_LR<n>. If GICH_LR<n>.HW == 1, indicating the INTID corresponds to a hardware interrupt, a deactivate request is also sent to the physical

Distributor, identifying the physical INTID from the corresponding field in the List register. This effect is identical to a Non-secure write to GICC_DIR from the PE having that physical INTID. This means that if the corresponding physical interrupt is marked as Group 0, and GICD_CTLR.DS == 0, the deactivation request is ignored. See GICC_EOIR for more information.

Note

Only Group 1 interrupts can target the hypervisor, and therefore only Group 1 interrupts are deactivated in the Distributor.

Accessing GICV_EOIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_EOIR0 provides equivalent functionality.
- For AArch64 implementations, ICC_EOIR0_EL1 provides equivalent functionality.

This register is used for Group 0 interrupts only. GICV_AEOIR provides equivalent functionality for Group 1 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_EOIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x0010	GICV_EOIR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **WO**.
- When an access is Secure, accesses to this register are **WO**.
- When an access is Non-secure, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_HPPIR, Virtual Machine Highest Priority Pending Interrupt Register

The GICV_HPPIR characteristics are:

Purpose

Provides the INTID of the highest priority pending Group 0 virtual interrupt in the List registers.

This register corresponds to the physical CPU interface register GICC_HPPIR.

Configuration

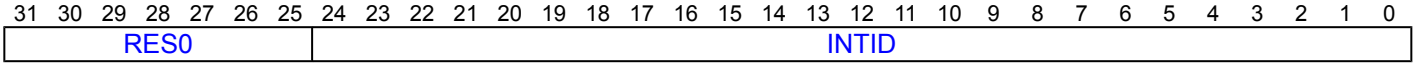
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_HPPIR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_HPPIR is a 32-bit register.

Field descriptions



Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Additional information

Reads of the GICC_HPPIR that do not return a valid INTID return a spurious INTID, 1022 or 1023. See 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Highest priority pending interrupt Group	GICV_HPPIR read	GICV_CTLR.AckCtl	Returned INTID
1	Non-secure	x	ID of Group 1 interrupt
1	Secure	0	1022
1	Secure	1	ID of Group 1 interrupt
0	Non-secure	x	1023
0	Secure	x	ID of Group 0 interrupt
No pending interrupts	x	x	1023

If the CPU interface supports only a single Security state, the entries that apply to Secure reads describe the behavior.

Accessing GICV_HPPIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_HPPIR0 provides equivalent functionality.
- For AArch64 implementations, ICC_HPPIR0_EL1 provides equivalent functionality.

This register is used for Group 0 interrupts only. GICV_AHPPIR provides equivalent functionality for Group 1 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_HPPIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x0018	GICV_HPPIR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_IAR, Virtual Machine Interrupt Acknowledge Register

The GICV_IAR characteristics are:

Purpose

Provides the INTID of the signaled Group 0 virtual interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

This register corresponds to the physical CPU interface register GICC_IAR.

Configuration

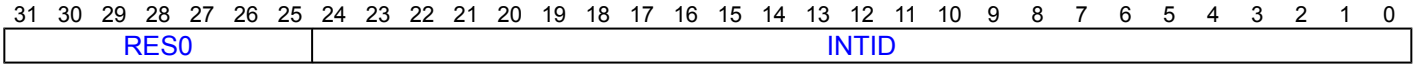
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_IAR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_IAR is a 32-bit register.

Field descriptions



Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Additional information

When the virtual machine writes to this register, the virtual CPU interface acknowledges the highest priority pending virtual interrupt and sets the state in the corresponding List register to active. The appropriate bit in the active priorities register GICH_APR<n> is set to 1.

If GICH_LR<n>.HW == 0, indicating that the interrupt is software-triggered, then bits [12:10] of GICH_LR<n> are returned in bits [12:10] of GICV_IAR. Otherwise bits [12:10] are RES0.

A read of this register returns the spurious INTID 1023 if either of the following is true:

- There are no pending interrupts of sufficiently high priority value to be signaled to the PE with the virtual CPU interface enabled and GICH_HCR.En == 1.
- Interrupt signaling by the virtual CPU interface is disabled.

A read of this register returns the spurious INTID 1022 if the highest priority pending interrupt is Group 1 and GICV_CTLR.AckCtl == 0.

Accessing GICV_IAR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_IAR0 provides equivalent functionality.
- For AArch64 implementations, ICC_IAR0_EL1 provides equivalent functionality.

This register is used for Group 0 interrupts only. GICV_AIAR provides equivalent functionality for Group 1 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_IAR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x000C	GICV_IAR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_IIDR, Virtual Machine CPU Interface Identification Register

The GICV_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the virtual CPU interface.

Configuration

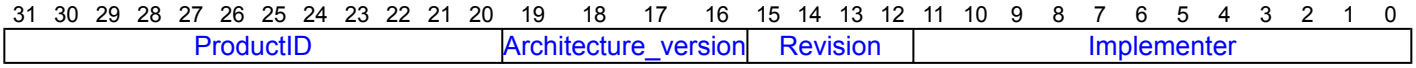
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_IIDR are RES0.

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states this register is Common.

Attributes

GICV_IIDR is a 32-bit register.

Field descriptions



ProductID, bits [31:20]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Architecture_version, bits [19:16]

The version of the GIC architecture that is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Architecture_version	Meaning
0b0001	GICv1.
0b0010	GICv2.
0b0011	GICv3 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1.
0b0100	GICv4 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1.

Other values are reserved.

Access to this field is **RO**.

Revision, bits [15:12]

Revision number for the CPU interface.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the CPU interface.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for GICV_IIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GICV_IIDR

GICV_IIDR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	`0x00FC`	GICV_IIDR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.
- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_PMR, Virtual Machine Priority Mask Register

The GICV_PMR characteristics are:

Purpose

This register provides a virtual interrupt priority filter. Only virtual interrupts with a higher priority than the value in this register are signaled to the PE.

Note

Higher interrupt priority corresponds to a lower value of the Priority field.

This register corresponds to the physical CPU interface register GICC_PMR.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_PMR are RES0.

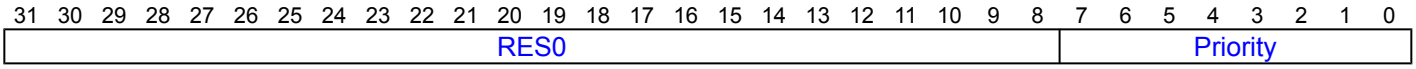
This register is available when the GIC implementation supports interrupt virtualization.

The Priority field of this register is aliased to GICH_VMCR.VMPR, to enable state to be switched easily between virtual machines during context-switching.

Attributes

GICV_PMR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the virtual CPU interface. If the priority of the interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

If the GIC implementation supports fewer than 256 priority levels some bits might be RAZ/WI, as follows:

- For 128 supported levels, bit [0] = `0b0`.
- For 64 supported levels, bits [1:0] = `0b00`.
- For 32 supported levels, bits [2:0] = `0b000`.
- For 16 supported levels, bits [3:0] = `0b0000`.

For more information, see 'Interrupt prioritization' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICV_PMR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_PMR provides equivalent functionality.
- For AArch64 implementations, ICC_PMR_EL1 provides equivalent functionality.

GICV_PMR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x0004	GICV_PMR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_RPR, Virtual Machine Running Priority Register

The GICV_RPR characteristics are:

Purpose

This register indicates the running priority of the virtual CPU interface.

This register corresponds to the physical CPU interface register GICC_RPR.

Configuration

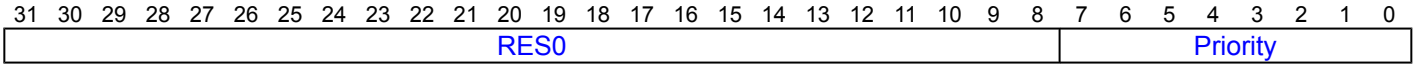
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_RPR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_RPR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the virtual CPU interface. This is the group priority of the current active interrupt.

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR was set to the minimum value.

Accessing GICV_RPR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_RPR provides equivalent functionality.
- For AArch64 implementations, ICC_RPR_EL1 provides equivalent functionality.

Depending on the implementation, if no bits are set to 1 in GICH_APR<n>, indicating no active virtual interrupts in the virtual CPU interface, the priority reads as `0xFF` or `0xF8` to reflect the number of supported interrupt priority bits defined by GICH_VTR.PRIbits.

GICV_RPR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	`0x0014`	GICV_RPR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RO**.

- When an access is Secure, accesses to this register are **RO**.
- When an access is Non-secure, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_STATUSR, Virtual Machine Error Reporting Status Register

The GICV_STATUSR characteristics are:

Purpose

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

Configuration

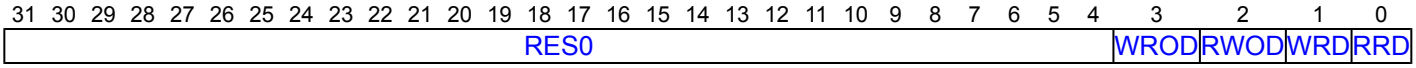
This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_STATUSR are RES0.

In systems where this register is implemented, Arm expects that when a virtual machine is scheduled, the hypervisor ensures that this register is cleared to 0. The hypervisor might check for illegal accesses when the virtual machine is unscheduled.

Attributes

GICV_STATUSR is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

WROD, bit [3]

Write to an RO location.

WROD	Meaning
0b0	Normal operation.
0b1	A write to an RO location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

RWOD, bit [2]

Read of a WO location.

RWOD	Meaning
0b0	Normal operation.
0b1	A read of a WO location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

WRD, bit [1]

Write to a reserved location.

WRD	Meaning
0b0	Normal operation.
0b1	A write to a reserved location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

RRD	Meaning
0b0	Normal operation.
0b1	A read of a reserved location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

Accessing GICV_STATUSR

This is an optional register. If the register is implemented, GICC_STATUSR must also be implemented. If the register is not implemented, the location is RAZ/WI.

This register is used only when System register access is not enabled. If System register access is enabled, this register is not updated. Equivalent function might be provided by appropriate traps and exceptions.

GICV_STATUSR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC Virtual CPU interface	0x002C	GICV_STATUSR

Accessible as follows:

- When GICD_CTLR.DS == 0, accesses to this register are **RW**.
- When an access is Secure, accesses to this register are **RW**.
- When an access is Non-secure, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_BASER<n>, ITS Table Descriptors, n = 0 - 7

The GITS_BASER<n> characteristics are:

Purpose

Specifies the base address and size of the ITS tables.

Configuration

A copy of this register is provided for each ITS table.

Bits [63:32] and bits [31:0] are accessible independently.

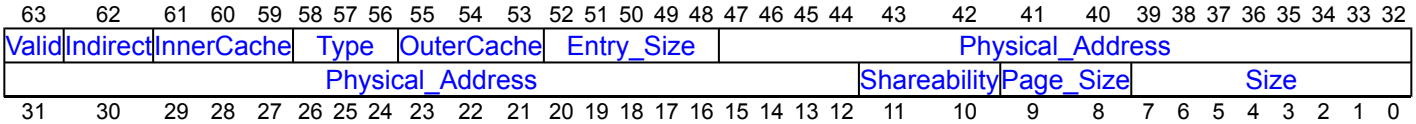
A maximum of 8 GITS_BASER<n> registers can be provided. Unimplemented registers are RES0.

When GITS_CTLR.Enabled == 1 or GITS_CTLR.Quiescent == 0, writing this register is UNPREDICTABLE.

Attributes

GITS_BASER<n> is a 64-bit register.

Field descriptions



Valid, bit [63]

Indicates whether software has allocated memory for the table:

Valid	Meaning
0b0	No memory is allocated for the table. The ITS discards any writes to the interrupt translation page when either: • GITS_BASER<n>.Type specifies any valid table entry type other than interrupt collections, that is, any value other than 0b100. • GITS_BASER<n>.Type specifies an interrupt collection and GITS_TYPER.HCC == 0.
0b1	Memory is allocated to the table.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Indirect, bit [62]

This field indicates whether an implemented register specifies a single, flat table or a two-level table where the first level contains a list of descriptors.

Indirect	Meaning
0b0	Single Level. The Size field indicates the number of pages used by the ITS to store data associated with each table entry.
0b1	Two Level. The Size field indicates the number of pages which contain an array of 64-bit descriptors to pages that are used to store the data associated with each table entry. A little endian memory order model is used.

For more information, see 'The ITS tables' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field is RAZ/WI for GIC implementations that only support flat tables. If the maximum width of the scaling factor that is identified by GITS_BASER<n>.Type and the smallest page size that is supported result in a single level table that requires multiple pages, then implementing this bit as RAZ/WI is DEPRECATED.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

InnerCache, bits [61:59]

Indicates the Inner Cacheability attributes of accesses to the table. The possible values of this field are:

InnerCache	Meaning
0b000	Device-nGnRnE.
0b001	Normal Inner Non-cacheable.
0b010	Normal Inner Cacheable Read-allocate, Write-through.
0b011	Normal Inner Cacheable Read-allocate, Write-back.
0b100	Normal Inner Cacheable Write-allocate, Write-through.
0b101	Normal Inner Cacheable Write-allocate, Write-back.
0b110	Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Type, bits [58:56]

Read only. Specifies the type of entity that requires entries in the corresponding table. The possible values of the field are:

Type	Meaning
0b000	Unimplemented. This register does not correspond to an ITS table.
0b001	Devices. This register corresponds to an ITS table that scales with the width of the DeviceID. Only a single GITS_BASER<n> register reports this type.
0b010	vPEs. FEAT_GICv4 only. This register corresponds to an ITS table that scales with the number of vPEs in the system. The table requires (ENTRY_SIZE * N) bytes of memory, where N is the number of vPEs in the system. Only a single GITS_BASER<n> register reports this type.
0b100	Interrupt collections. This register corresponds to an ITS table that scales with the number of interrupt collections in the system. The table requires (ENTRY_SIZE * N) bytes of memory, where N is the number of interrupt collections. Not more than one GITS_BASER<n> register will report this type.

Other values are reserved.

For FEAT_GICv4p1, the registers are allocated as follows:

- GITS_BASER0.Type is 0b001 (Device).
- GITS_BASER1.Type is either 0b100 (Collection Table) or 0b000 (Unimplemented).
- GITS_BASER2.Type is either 0b010 (vPE) or 0b000 (Unimplemented).
- GITS_BASER<n>.Type, where 'n' is in the range 3 to 7, is 0b000 (Unimplemented).

For FEAT_GICv3, FEAT_GICv3p1, and FEAT_GICv4, Arm recommends that the GITS_BASER<n> use the same allocations.

Other allocations of Type values are deprecated.

OuterCache, bits [55:53]

Indicates the Outer Cacheability attributes of accesses to the table. The possible values of this field are:

OuterCache	Meaning
0b000	Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.
0b001	Normal Outer Non-cacheable.
0b010	Normal Outer Cacheable Read-allocate, Write-through.
0b011	Normal Outer Cacheable Read-allocate, Write-back.
0b100	Normal Outer Cacheable Write-allocate, Write-through.
0b101	Normal Outer Cacheable Write-allocate, Write-back.
0b110	Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Entry_Size, bits [52:48]

Read-only. Specifies the number of bytes per table entry, minus one.

Physical_Address, bits [47:12]

Physical Address. When Page_Size is 4KB or 16KB:

- Bits [51:48] of the base physical address are zero.
- This field provides bits[47:12] of the base physical address of the table.
- Bits[11:0] of the base physical address are zero.
- The address must be aligned to the size specified in the Page Size field. Otherwise the effect is CONSTRAINED UNPREDICTABLE, and can be one of the following:
    - Bits[X:12], where X is derived from the page size, are treated as zero.
    - The value of bits[X:12] are used when calculating the address of a table access.

When Page_Size is 64KB:

- Bits[47:16] of the register provide bits[47:16] of the base physical address of the table.
- Bits[15:12] of the register provide bits[51:48] of the base physical address of the table.
- Bits[15:0] of the base physical address are 0.

In implementations that support fewer than 52 bits of physical address, any unimplemented upper bits might be RAZ/WI.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the table. The possible values of this field are:

Shareability	Meaning
0b00	Non-shareable.
0b01	Inner Shareable.
0b10	Outer Shareable.
0b11	Reserved. Treated as 0b00.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Page_Size, bits [9:8]

The size of page that the table uses:

Page_Size	Meaning
0b00	4KB.
0b01	16KB.
0b10	64KB.
0b11	Reserved. Treated as 0b10.

Note

If the GIC implementation supports only a single, fixed page size, this field might be RO.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Size, bits [7:0]

The number of pages of physical memory allocated to the table, minus one. GITS_BASER<n>.Page_Size specifies the size of each page.

If GITS_BASER<n>.Type == 0, this field is RAZ/WI.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GITS_BASER<n>

GITS_BASER<n> can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC ITS control	0x0100 + (8 * n)	GITS_BASER<n>

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_CBASER, ITS Command Queue Descriptor

The GITS_CBASER characteristics are:

Purpose

Specifies the base address and size of the ITS command queue.

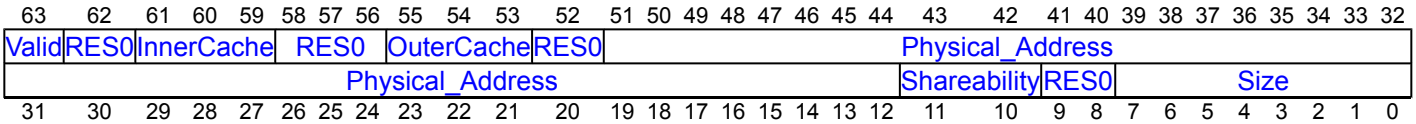
Configuration

Bits [63:32] and bits [31:0] are accessible separately.

Attributes

GITS_CBASER is a 64-bit register.

Field descriptions



Valid, bit [63]

Indicates whether software has allocated memory for the command queue:

Valid	Meaning
0b0	No memory is allocated for the command queue.
0b1	Memory is allocated to the command queue.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Bit [62]

Reserved, RES0.

InnerCache, bits [61:59]

Indicates the Inner Cacheability attributes of accesses to the command queue. The possible values of this field are:

InnerCache	Meaning
0b000	Device-nGnRnE.
0b001	Normal Inner Non-cacheable.
0b010	Normal Inner Cacheable Read-allocate, Write-through.
0b011	Normal Inner Cacheable Read-allocate, Write-back.
0b100	Normal Inner Cacheable Write-allocate, Write-through.
0b101	Normal Inner Cacheable Write-allocate, Write-back.
0b110	Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [58:56]

Reserved, RES0.

OuterCache, bits [55:53]

Indicates the Outer Cacheability attributes of accesses to the command queue. The possible values of this field are:

OuterCache	Meaning
0b000	Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.
0b001	Normal Outer Non-cacheable.
0b010	Normal Outer Cacheable Read-allocate, Write-through.
0b011	Normal Outer Cacheable Read-allocate, Write-back.
0b100	Normal Outer Cacheable Write-allocate, Write-through.
0b101	Normal Outer Cacheable Write-allocate, Write-back.
0b110	Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.
0b111	Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bit [52]

Reserved, RES0.

Physical_Address, bits [51:12]

Bits [51:12] of the base physical address of the command queue. Bits [11:0] of the base address are 0.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

If bits [15:12] are not all zeros, behavior is a CONSTRAINED UNPREDICTABLE choice:

- Bits [15:12] are treated as if all the bits are zero. The value read back from those bits is either the value written or zero.
- The result of the calculation of an address for a command queue read can be corrupted.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the command queue. The possible values of this field are:

Shareability	Meaning
0b00	Non-shareable.
0b01	Inner Shareable.
0b10	Outer Shareable.
0b11	Reserved. Treated as 0b00.

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [9:8]

Reserved, RES0.

Size, bits [7:0]

The number of 4KB pages of physical memory allocated to the command queue, minus one.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Additional information

The command queue is a circular buffer and wraps at Physical Address [47:0] + (4096 * (Size + 1)).

Note

When this register is successfully written, the value of GITS_CREADR is set to zero.

Accessing GITS_CBASER

When GITS_CTLR.Enabled == 1 or GITS_CTLR.Quiescent == 0, writing this register is UNPREDICTABLE.

GITS_CBASER can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC ITS control	0x0080	GITS_CBASER

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_CREADR, ITS Read Register

The GITS_CREADR characteristics are:

Purpose

Specifies the offset from GITS_CBASER where the ITS reads the next ITS command.

Configuration

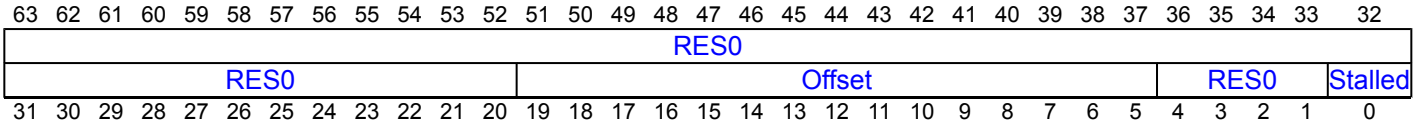
This register is cleared to 0 when a value is written to GITS_CBASER.

Bits [63:32] and bits [31:0] are accessible separately.

Attributes

GITS_CREADR is a 64-bit register.

Field descriptions



Bits [63:20]

Reserved, RES0.

Offset, bits [19:5]

Bits [19:5] of the offset from GITS_CBASER. Bits [4:0] of the offset are zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [4:1]

Reserved, RES0.

Stalled, bit [0]

Reports whether the processing of commands is stalled because of a command error.

Stalled	Meaning
0b0	ITS command queue is not stalled because of a command error.
0b1	ITS command queue is stalled because of a command error.

Additional information

For more information, see 'The ITS command interface' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accessing GITS_CREADR

GITS_CREADR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC ITS control	0x0090	GITS_CREADR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_CTLR, ITS Control Register

The GITS_CTLR characteristics are:

Purpose

Controls the operation of an ITS.

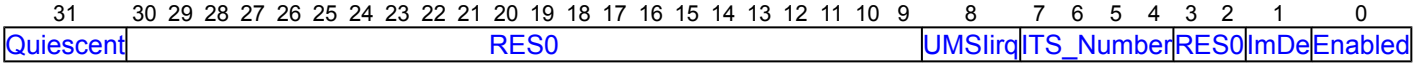
Configuration

The ITS_Number (bits [7:4]) and bit [1] fields apply only in FEAT_GICv4 implementations, and are RES0 in FEAT_GICv3 implementations.

Attributes

GITS_CTLR is a 32-bit register.

Field descriptions



Quiescent, bit [31]

Read-only. Indicates completion of all ITS operations when GITS_CTLR.Enabled == 0.

Quiescent	Meaning
0b0	The ITS is not quiescent and cannot be powered down.
0b1	The ITS is quiescent and can be powered down.

For the ITS to be considered inactive, there must be no transactions in progress. In addition, all operations required to ensure that mapping data is consistent with external memory must be complete.

Note

In distributed GIC implementations, this bit is set to 1 only after the ITS forwards any operations that have not yet been completed to the Redistributors and receives confirmation that all such operations have reached the appropriate Redistributor.

In FEAT_GICv3, FEAT_GICv3p1, and FEAT_GICv4, when GITS_CTLR.Enabled == 1, the value of GITS_CTLR.Quiescent is UNKNOWN.

In FEAT_GICv4p1, when GITS_CTLR.Enabled == 1, the value of GITS_CTLR.Quiescent reads as 1 until the write to Enabled has taken effect and then reads as 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to '1'.

Bits [30:9]

Reserved, RES0.

UMSIirq, bit [8]

Unmapped MSI reporting interrupt enable.

UMSIirq	Meaning
0b0	The ITS does not assert an interrupt signal when GITS_STATUSR.UMSI is 1.
0b1	The ITS asserts an interrupt signal when GITS_STATUSR.UMSI is 1.

If GITS_TYPER.UMSIirq is 0, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

ITS_Number, bits [7:4]

In FEAT_GICv3 implementations this field is RES0.

In FEAT_GICv4 implementations with more than one ITS instance, this field indicates the ITS number for use with 'VMOVP GICv4.0' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

When GITS_TYPER.VMOVP is 1, this field may be implemented as RES0.

If this field is programmable, changing this field when GITS_CTLR.Quiescent == 0 or GITS_CTLR.Enabled == 1 is UNPREDICTABLE.

It is IMPLEMENTATION DEFINED whether this field is programmable or RO.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [3:2]

Reserved, RES0.

ImDe, bit [1]

In GICv3 implementations, this bit is RES0.

In GICv4 implementations, this bit is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Enabled, bit [0]

Controls whether the ITS is enabled:

Enabled	Meaning
0b0	The ITS is not enabled. Writes to GITS_TRANSLATER are ignored and no further command queue entries are processed.
0b1	The ITS is enabled. Writes to GITS_TRANSLATER result in interrupt translations and the command queue is processed.

If a write to this register changes this field from 1 to 0, the ITS must ensure that both:

- Any caches containing mapping data are made consistent with external memory.
- GITS_CTLR.Quiescent == 0 until all caches are consistent with external memory.

Changing GITS_CTLR.Enabled from 0 to 1 when GITS_CTLR.Quiescent is 0 results in UNPREDICTABLE behavior.

The reset behavior of this field is:

- On a GIC reset, this field resets to '0'.

Accessing GITS_CTLR

GITS_CTLR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC ITS control	0x0000	GITS_CTLR

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_CWRITER, ITS Write Register

The GITS_CWRITER characteristics are:

Purpose

Specifies the offset from GITS_CBASER where software writes the next ITS command.

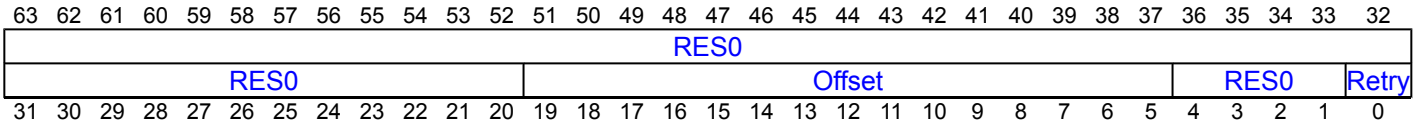
Configuration

Bits [63:32] and bits [31:0] are accessible separately.

Attributes

GITS_CWRITER is a 64-bit register.

Field descriptions



Bits [63:20]

Reserved, RES0.

Offset, bits [19:5]

Bits [19:5] of the offset from GITS_CBASER. Bits [4:0] of the offset are zero.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [4:1]

Reserved, RES0.

Retry, bit [0]

Writing this bit has the following effects:

Retry	Meaning
0b0	No effect on the processing commands by the ITS.
0b1	Restarts the processing of commands by the ITS if it stalled because of a command error. **Note** If the processing of commands is not stalled because of a command error, writing 1 to this bit has no effect.

When read, this bit is RES0.

Additional information

For more information, see 'The ITS command interface' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

If GITS_CWRITER is written with a value outside of the valid range specified by GITS_CBASER.Physical_Address and GITS_CBASER.Size, behavior is a CONSTRAINED UNPREDICTABLE choice, as follows:

- The command queue is considered invalid, and no further commands are processed until GITS_CWRITER is written with a value that is in the valid range.
- The value is treated as a valid UNKNOWN value.

An implementation might choose to report a system error in an IMPLEMENTATION DEFINED manner.

Accessing GITS_CWRITER

GITS_CWRITER can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC ITS control	0x0088	GITS_CWRITER

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_IIDR, ITS Identification Register

The GITS_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the ITS.

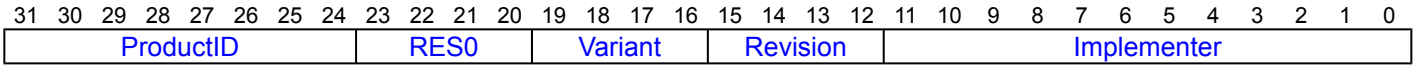
Configuration

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

Attributes

GITS_IIDR is a 32-bit register.

Field descriptions



ProductID, bits [31:24]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

Variant, bits [19:16]

Variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Revision number. Typically, this field is used to distinguish minor revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the ITS.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for GITS_IIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GITS_IIDR

GITS_IIDR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC ITS control	`0x0004`	GITS_IIDR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_MPAMIDR, Report maximum PARTID and PMG Register

The GITS_MPAMIDR characteristics are:

Purpose

Reports the maximum support PARTID and PMG values.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GITS_MPAMIDR are RES0.

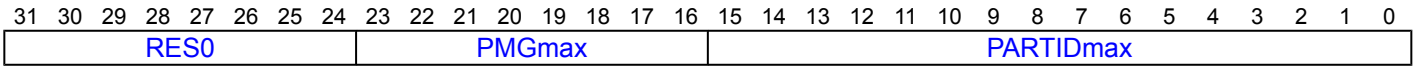
A copy of this register is provided for each ITS.

When GITS_TYPER.MPAM==0, this register is RES0.

Attributes

GITS_MPAMIDR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

PMGmax, bits [23:16]

Maximum PMG value supported.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PARTIDmax, bits [15:0]

Maximum PARTID value supported.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing GITS_MPAMIDR

GITS_MPAMIDR can be accessed through the memory-mapped interfaces:

Component	Offset
GIC ITS control	0x0010

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_MPIDR, Report ITS's affinity.

The GITS_MPIDR characteristics are:

Purpose

Reports ITS's affinity when the vPE Table is shared with Redistributors.

Configuration

This register is present only when GICv4.1 is implemented. Otherwise, direct accesses to GITS_MPIDR are RES0.

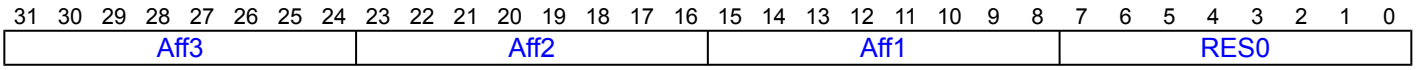
A copy of this register is provided for each ITS.

When GITS_TYPER.SVPET==0, this register is RES0.

Attributes

GITS_MPIDR is a 32-bit register.

Field descriptions



Aff3, bits [31:24]

The Affinity level 3 value for the ITS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff2, bits [23:16]

The Affinity level 2 value for the ITS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

The Affinity level 1 value for the ITS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [7:0]

Reserved, RES0.

Accessing GITS_MPIDR

GITS_MPIDR can be accessed through the memory-mapped interfaces:

Component	Offset
GIC ITS control	0x0018

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_PARTIDR, Set PARTID and PMG Register

The GITS_PARTIDR characteristics are:

Purpose

Sets the PARTID and PMG values used for memory accesses by the ITS.

Configuration

This register is present only when GICv3.1 is implemented. Otherwise, direct accesses to GITS_PARTIDR are RES0.

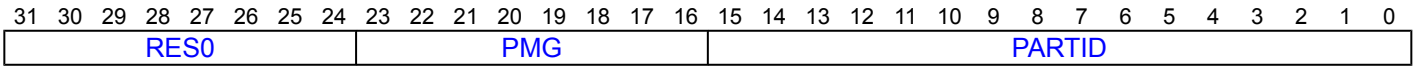
A copy of this register is provided for each ITS.

When GITS_TYPER.MPAM==0, this register is RES0.

Attributes

GITS_PARTIDR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

PMG value used when ITS accesses memory.

Bits not needed to represent PMG values in the range 0 to PMG_MAX are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'00000000'`.

PARTID, bits [15:0]

PARTID value used when ITS accesses memory.

Bits not needed to represent PARTID values in the range 0 to PARTID_MAX are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to `'0000000000000000'`.

Accessing GITS_PARTIDR

GITS_PARTIDR can be accessed through the memory-mapped interfaces:

Component	Offset
GIC ITS control	0x0014

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_SGIR, ITS SGI Register

The GITS_SGIR characteristics are:

Purpose

Written by software to signal a virtual SGI for translation by the ITS.

Configuration

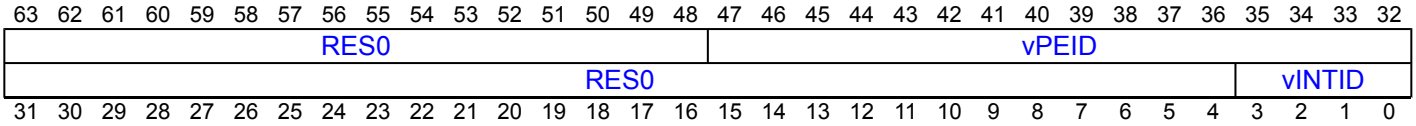
This register is present only when GICv4.1 is implemented. Otherwise, direct accesses to GITS_SGIR are RES0.

This register is provided only in FEAT_GICv4p1 implementations.

Attributes

GITS_SGIR is a 64-bit register.

Field descriptions



Bits [63:48]

Reserved, RES0.

vPEID, bits [47:32]

ID of target vPEID.

The size of this field is IMPLEMENTATION DEFINED, and is specified by the GICD_TYPER2.VIL and GICD_TYPER2.VID fields. Unimplemented bits are RES0.

Bits [31:4]

Reserved, RES0.

vINTID, bits [3:0]

INTID of virtual SGI.

Accessing GITS_SGIR

64-bit access only.

GITS_SGIR can be accessed through the memory-mapped interfaces:

Component	Offset
GIC ITS control	0x20020

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_STATUSR, ITS Error Reporting Status Register

The GITS_STATUSR characteristics are:

Purpose

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.
- Unmapped MSIs.

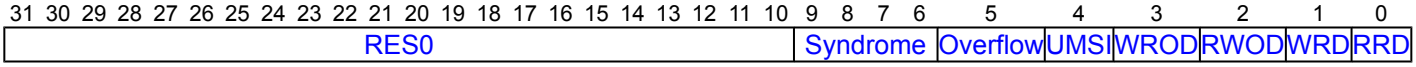
Configuration

There are no configuration notes.

Attributes

GITS_STATUSR is a 32-bit register.

Field descriptions



Bits [31:10]

Reserved, RES0.

Syndrome, bits [9:6]

Syndrome for the MSI that set GITS_STATUSR.UMSI to 1.

Syndrome	Meaning
0b0000	Unknown reason.
0b0010	DeviceID out of range.
0b0011	DeviceID unmapped.
0b0100	EventID out of range.
0b0101	EventID unmapped.
0b0111	Collection unmapped.
0b1001	vPEID unmapped.

An implementation might not support reporting all syndromes, and might report 0b0000 for any cause.

This field is UNKNOWN when GITS_STATUSR.UMSI is 0.

Overflow, bit [5]

Reports whether an unmapped MSI has been received while GITS_STATUSR.UMSI is 1.

Overflow	Meaning
0b0	No unmapped MSIs have been received since GITS_STATUSR.UMSI set to 1.
0b1	At least one unmapped MSIs have been received since GITS_STATUSR.UMSI set to 1.

A software write of 1 to the bit clears it. A write of any other value is ignored.

If GITS_TYPER.UMSI is 0, this field is RES0.

UMSI, bit [4]

Reports whether an unmapped MSI has been received

An unmapped MSI is defined as an MSI arriving at GITS_TRANSLATER for which there is insufficient mapping information for it to be forwarded to a Redistributor.

It is IMPLEMENTATION DEFINED whether an INT command can be reported as an unmapped MSI.

UMSI	Meaning
0b0	No unmapped MSIs have been received.
0b1	Unmapped MSI received.

A software write of 1 to the bit clears it. A write of any other value is ignored.

If GITS_TYPER.UMSI is 0, this field is RES0.

WROD, bit [3]

Write to an RO location.

WROD	Meaning
0b0	Normal operation.
0b1	A write to an RO location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

RWOD, bit [2]

Read of a WO location.

RWOD	Meaning
0b0	Normal operation.
0b1	A read of a WO location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

WRD, bit [1]

Write to a reserved location.

WRD	Meaning
0b0	Normal operation.
0b1	A write to a reserved location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

RRD	Meaning
0b0	Normal operation.
0b1	A read of a reserved location has been detected.

When a violation is detected, software must write 1 to this register to reset it.

Accessing GITS_STATUSR

This is an optional register. If the register is not implemented, the location is RAZ/WI.

GITS_STATUSR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC ITS control	0x0040	GITS_STATUSR

Accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_TRANSLATER, ITS Translation Register

The GITS_TRANSLATER characteristics are:

Purpose

Written by a requesting Device to signal an interrupt for translation by the ITS.

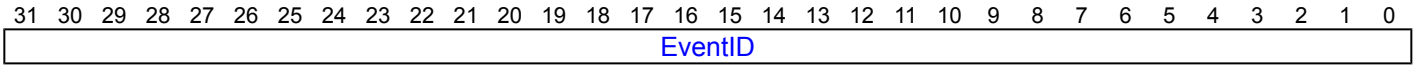
Configuration

This register is at the same offset as GICD_SETSPI_NSR in the Distributor, and is at the same offset as GICR_SETLPIR in the Redistributor.

Attributes

GITS_TRANSLATER is a 32-bit register.

Field descriptions



EventID, bits [31:0]

An identifier corresponding to the interrupt to be translated.

Note

The size of the EventID is DeviceID specific, and set when the DeviceID is mapped to an ITT (using MAPD).

The number of EventID bits implemented is reported by GITS_TYPER.ID_bits. If a write specifies nonzero identifiers bits outside this range behavior is a CONSTRAINED UNPREDICTABLE choice between:

- Nonzero identifier bits outside the supported range are ignored.
- The write is ignored.

Additional information

The DeviceID presented to an ITS is used to index a device table. The device table maps the DeviceID to an interrupt translation table for that device.

Accessing GITS_TRANSLATER

16-bit access to bits [15:0] of this register must be supported. When this register is written by a 16-bit transaction, bits [31:16] are written as zero.

Implementations must ensure that:

- A unique DeviceID is provided for each requesting device, and the DeviceID is presented to the ITS when a write to this register occurs in a manner that cannot be spoofed by any agent capable of performing writes.
- The DeviceID presented corresponds to the DeviceID field in the ITS commands.

Writes to this register are ignored if any of the following are true:

- GITS_CTLR.Enabled == 0.
- The presented DeviceID is not mapped to an Interrupt Translation Table.
- The DeviceID is larger than the supported size.
- The DeviceID is mapped to an Interrupt Translation Table, but the EventID is outside the range specified by MAPD.

- The EventID is mapped to an Interrupt Translation Table and the EventID is within the range specified by MAPD, but the EventID is unmapped.

Translation requests that result from writes to this register are subject to certain ordering rules. For more information, see 'Ordering of translations with the output to ITS commands' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

GITS_TRANSLATER can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC ITS translation	0x0040	GITS_TRANSLATER

Accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_TYPER, ITS Type Register

The GITS_TYPER characteristics are:

Purpose

Specifies the features that an ITS supports.

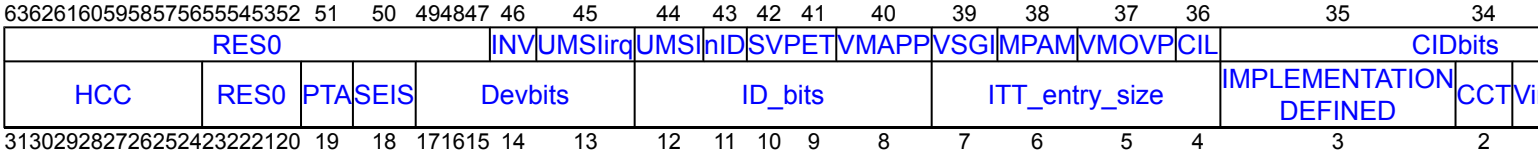
Configuration

There are no configuration notes.

Attributes

GITS_TYPER is a 64-bit register.

Field descriptions



Bits [63:47]

Reserved, RES0.

INV, bit [46]

ITS cache invalidation behavior on disable.

The value of this field is an IMPLEMENTATION DEFINED choice of:

INV	Meaning
0b0	It is IMPLEMENTATION DEFINED whether ITS caches are invalidated on clearing GITS_CTLR.Enabled and GITS_BASER<n>.Valid.
0b1	ITS caches are invalidated on clearing GITS_CTLR.Enabled and GITS_BASER<n>.Valid.

If GITS_TYPER.INV is 1, after the following sequence:

- GITS_CTLR.Enabled written to 0.
- A read of GITS_CTLR.Quiescent returns 1.
- GITS_BASER<n>.Valid written to 0.

There is no cached information from the ITS memory structure pointed to by GITS_BASER<n>.

Access to this field is **RO**.

UMSIirq, bit [45]

Indicates support for generating an interrupt on receiving unmapped MSI.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UMSIirq	Meaning
0b0	Interrupt on unmapped MSI not supported.
0b1	Interrupt on unmapped MSI is supported.

If GITS_TYPER.UMSI is 0, this field is RES0.

Access to this field is **RO**.

UMSI, bit [44]

Indicates support for reporting receipt of unmapped MSIs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UMSI	Meaning
0b0	Reporting of unmapped MSIs is not supported.
0b1	Reporting of unmapped MSIs is supported.

Access to this field is **RO**.

nID, bit [43]

When GICv4.1 is implemented:

nID

The value of this field is an IMPLEMENTATION DEFINED choice of:

nID	Meaning
0b0	Individual doorbell interrupt supported.
0b1	Individual doorbell interrupt not supported.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

SVPET, bits [42:41]

When GICv4.1 is implemented:

SVPET

The value of this field is an IMPLEMENTATION DEFINED choice of:

SVPET	Meaning
0b00	vPE Table is not shared with Redistributors.
0b01	vPE Table is shared with the groups of Redistributors indicated by GITS_MPIDR.Aff3.
0b10	vPE Table is shared with the groups of Redistributors indicated by GITS_MPIDR fields Aff3 and Aff2.
0b11	vPE Table is shared with the groups of Redistributors indicated by GITS_MPIDR fields Aff3, Aff2 and Aff1.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

VMAPP, bit [40]

When GICv4.1 is implemented:

VMAPP

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMAPP	Meaning
0b0	FEAT_GICv4 VMAPP command layout.
0b1	FEAT_GICv4p1 VMAPP command layout.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

VSGI, bit [39]

When GICv4.1 is implemented:

VSGI

The value of this field is an IMPLEMENTATION DEFINED choice of:

VSGI	Meaning
0b0	Direct injection of SGIs is not supported.
0b1	Direct injection of SGIs is supported.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

MPAM, bit [38]

When GICv3.1 is implemented:

MPAM

The value of this field is an IMPLEMENTATION DEFINED choice of:

MPAM	Meaning
0b0	MPAM is not supported.
0b1	MPAM is supported.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

VMOVP, bit [37]

When GICv4 is implemented:

Indicates the form of the VMOVP command.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMOVP	Meaning
0b0	When moving a vPE, software must issue a VMOVP on all ITSs that have mappings for that vPE. The ITSList and Sequence Number fields in the VMOVP command must ensure synchronization, otherwise behavior is UNPREDICTABLE.
0b1	When moving a vPE, software must only issue a VMOVP on one of the ITSs that has a mapping for that vPE. The ITSList and Sequence Number fields in the VMOVP command are RES0.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

CIL, bit [36]

Collection ID Limit.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CIL	Meaning
0b0	ITS supports 16-bit Collection ID, GITS_TYPER.CIDbits is RES0.
0b1	GITS_TYPER.CIDbits indicates supported Collection ID size

In implementations that do not support Collections in external memory, this bit is RES0 and the number of Collections supported is reported by GITS_TYPER.HCC.

Access to this field is **RO**.

CIDbits, bits [35:32]

Number of Collection ID bits.

- The number of bits of Collection ID minus one.
- When GITS_TYPER.CIL == 0, this field is RES0.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

HCC, bits [31:24]

Hardware Collection Count. The number of interrupt collections supported by the ITS without provisioning of external memory.

Note

Collections held in hardware are unmapped at reset.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

PTA, bit [19]

Physical Target Addresses. Indicates the format of the target address:

The value of this field is an IMPLEMENTATION DEFINED choice of:

PTA	Meaning
0b0	The target address corresponds to the PE number specified by GICR_TYPER.Processor_Number.
0b1	The target address corresponds to the base physical address of the required Redistributor.

For more information, see 'RDbase' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Access to this field is **RO**.

SEIS, bit [18]

SEI support. Indicates whether the virtual CPU interface supports generation of SEIs:

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEIS	Meaning
0b0	The ITS does not support local generation of SEIs.
0b1	The ITS supports local generation of SEIs.

Access to this field is **RO**.

Devbits, bits [17:13]

The number of DeviceID bits implemented, minus one.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ID_bits, bits [12:8]

The number of EventID bits implemented, minus one.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ITT_entry_size, bits [7:4]

Read-only. Indicates the number of bytes per translation table entry, minus one.

For more information about the ITS command 'MAPD', see MAPD.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

IMPLEMENTATION DEFINED, bit [3]

IMPLEMENTATION DEFINED.

CCT, bit [2]

Cumulative Collection Tables.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCT	Meaning
0b0	The total number of supported collections is determined by the number of collections held in memory only.
0b1	The total number of supported collections is determined by number of collections that are held in memory and the number indicated by GITS_TYPER.HCC.

If GITS_TYPER.HCC == 0, or if memory backed collections are not supported (all GITS_BASER<n>.Type != 100), this bit is RES0.

Access to this field is **RO**.

Virtual, bit [1]

When GICv4 is implemented:

Indicates whether the ITS supports virtual LPIs and direct injection of virtual LPIs:

The value of this field is an IMPLEMENTATION DEFINED choice of:

Virtual	Meaning
0b0	The ITS does not support virtual LPIs or direct injection of virtual LPIs.
0b1	The ITS supports virtual LPIs and direct injection of virtual LPIs.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Physical, bit [0]

Indicates whether the ITS supports physical LPIs:

The value of this field is an IMPLEMENTATION DEFINED choice of:

Physical	Meaning
0b0	The ITS does not support physical LPIs.
0b1	The ITS supports physical LPIs.

This field is RES1, indicating that the ITS supports physical LPIs.

Access to this field is **RO**.

Accessing GITS_TYPER

GITS_TYPER can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC ITS control	0x0008	GITS_TYPER

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_UMSIR, ITS Unmapped MSI register

The GITS_UMSIR characteristics are:

Purpose

Provides the DeviceID and EventID of the unmapped MSI that set GITS_STATUSR.UMSI.

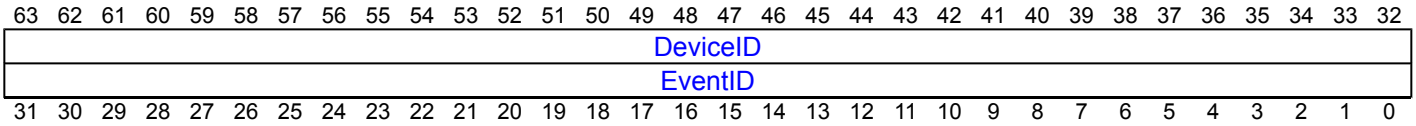
Configuration

This register is present only when GITS_TYPER.UMSI == 1. Otherwise, direct accesses to GITS_UMSIR are RES0.

Attributes

GITS_UMSIR is a 64-bit register.

Field descriptions



DeviceID, bits [63:32]

DeviceID of MSI that set GITS_STATUSR.UMSI to 1.

If GITS_STATUSR.UMSI is 0, this field is UNKNOWN.

EventID, bits [31:0]

EventID of MSI that set GITS_STATUSR.UMSI to 1.

If GITS_STATUSR.UMSI is 0, this field is UNKNOWN.

Accessing GITS_UMSIR

GITS_UMSIR can be accessed through the memory-mapped interfaces:

Component	Offset	Instance
GIC ITS control	0x0048	GITS_UMSIR

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MIDR_EL1, Main ID Register

The MIDR_EL1 characteristics are:

Purpose

Provides identification information for the PE, including an implementer code for the device and a device ID number.

Configuration

External register MIDR_EL1 bits [31:0] are architecturally mapped to AArch64 System register MIDR_EL1[31:0].

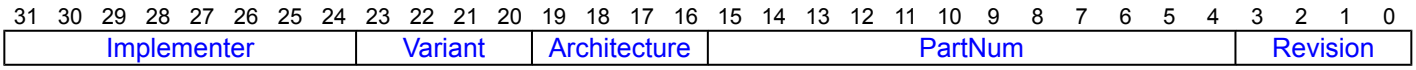
External register MIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MIDR[31:0].

The power domain of MIDR_EL1 is IMPLEMENTATION DEFINED.

Attributes

MIDR_EL1 is a 32-bit register.

Field descriptions



Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Implementer	Meaning
0x00	Reserved for software use.
0x41	Arm Limited.
0x42	Broadcom Corporation.
0x43	Cavium Inc.
0x44	Digital Equipment Corporation.
0x46	Fujitsu Ltd.
0x49	Infineon Technologies AG.
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation.
0x50	Applied Micro Circuits Corporation.
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation.
0xC0	Ampere Computing.

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

Access to this field is **RO**.

Variant, bits [23:20]

Variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Architecture, bits [19:16]

Architecture version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Architecture	Meaning
0b0001	Armv4.
0b0010	Armv4T.
0b0011	Armv5 (obsolete).
0b0100	Armv5T.
0b0101	Armv5TE.
0b0110	Armv5TEJ.
0b0111	Armv6.
0b1111	Architectural features are individually identified in the ID_* registers.

All other values are reserved.

Access to this field is **RO**.

PartNum, bits [15:4]

Primary Part Number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [3:0]

Revision number for the device.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MIDR_EL1

MIDR_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0xD00	MIDR_EL1

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register are **IMPDEF**.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_CASSOC, MPAM Cache Maximum Associativity Partition Configuration Register

The MPAMCFG_CASSOC characteristics are:

Purpose

The MPAMCFG_CASSOC is a 32-bit read/write register that controls the maximum fraction of the cache associativity that the PARTID selected by MPAMCFG_PART_SEL is permitted to allocate.

MPAMCFG_CASSOC_s controls the cache maximum associativity for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_CASSOC_ns controls the cache maximum associativity for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. MPAMCFG_CASSOC_rl controls the cache maximum associativity for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL. MPAMCFG_CASSOC_rt controls the cache maximum associativity for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configuration

The power domain of MPAMCFG_CASSOC is IMPLEMENTATION DEFINED.

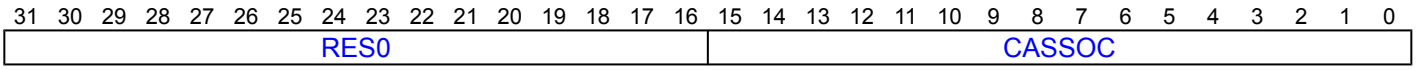
This register is present only when MPAMF_IDR.HAS_CCAP_PART == 1, (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented), and MPAMF_CCAP_IDR.HAS_CASSOC == 1. Otherwise, direct accesses to MPAMCFG_CASSOC are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_CASSOC is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

CASSOC, bits [15:0]

Maximum cache associativity usage in fixed-point fraction format by the partition selected by MPAMCFG_PART_SEL. The fraction represents the portion of the cache associativity that the PARTID is permitted to allocate. CASSOC controls the fraction of associativity in each associativity grouping of the cache. In a set associative cache, CASSOC applies to the fraction of the ways in each set.

The implemented width of the fixed-point fraction is given in MPAMF_CCAP_IDR.CASSOC_WD. Unimplemented bits within the field are RAZ/WI. The implemented bits of the CASSOC field are always the most significant bits of the field.

The fixed-point fraction CASSOC is less than 1. The implied binary point is between bits 15 and 16. In an implementation with w implemented bits, the largest fraction of the cache that can be represented is $1- (0.5)^{W}$.

Accessing MPAMCFG_CASSOC

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_CASSOC_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_CASSOC_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_CASSOC_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_CASSOC_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_CASSOC_s, MPAMCFG_CASSOC_ns, MPAMCFG_CASSOC_rt, and MPAMCFG_CASSOC_rl must be separate registers:

- The Secure instance (MPAMCFG_CASSOC_s) accesses the cache maximum associativity partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_CASSOC_ns) accesses the cache maximum associativity partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_CASSOC_rt) accesses the cache maximum associativity partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_CASSOC_rl) accesses the cache maximum associativity partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_CASSOC access the cache maximum associativity partitioning configuration settings for the cache resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_CASSOC access the cache maximum associativity partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_CASSOC access the cache maximum associativity partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_CASSOC access the cache maximum associativity partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_CASSOC can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0118	MPAMCFG_CASSOC_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0118	MPAMCFG_CASSOC_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0118	MPAMCFG_CASSOC_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0118	MPAMCFG_CASSOC_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_CMAX, MPAM Cache Maximum Capacity Partition Configuration Register

The MPAMCFG_CMAX characteristics are:

Purpose

The MPAMCFG_CMAX is a 32-bit read/write register that controls the maximum fraction of the cache capacity that the PARTID selected by MPAMCFG_PART_SEL is permitted to allocate.

MPAMCFG_CMAX_s controls the cache maximum capacity for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_CMAX_ns controls the cache maximum capacity for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. MPAMCFG_CMAX_rt controls the cache maximum capacity for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_CMAX_rl controls the cache maximum capacity for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configuration

The power domain of MPAMCFG_CMAX is IMPLEMENTATION DEFINED.

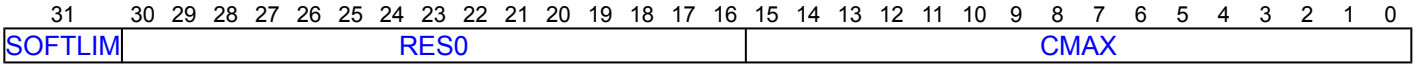
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_CCAP_PART == 1, and MPAMF_CCAP_IDR.NO_CMAX == 0. Otherwise, direct accesses to MPAMCFG_CMAX are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_CMAX is a 32-bit register.

Field descriptions



SOFTLIM, bit [31]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_CCAP_IDR.HAS_CMAX_SOFTLIM == 1:

Soft limiting of CMAX. Soft limiting allows some allocations by a PARTID when its cache use is above the CMAX maximum cache capacity.

SOFTLIM	Meaning
0b0	When CMAX cache capacity is exceeded, the partition is not allowed to increase its cache capacity usage. It is only permitted to replace a line that was previously occupied by a line allocated by that PARTID.
0b1	When CMAX cache capacity is exceeded, the partition is permitted to allocate capacity beyond CMAX, but only from invalid lines or lines belonging to disabled PARTIDs.

Otherwise:

Reserved, RES0.

Bits [30:16]

Reserved, RES0.

CMAX, bits [15:0]

Maximum cache capacity usage in fixed-point fraction format by the partition selected by MPAMCFG_PART_SEL. The fraction represents the portion of the total cache capacity that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in MPAMF_CCAP_IDR.CMAX_WD. Unimplemented bits within the field are RAZ/WI. The implemented bits of the CMAX field are always the most significant bits of the field.

The fixed-point fraction CMAX is less than 1. The implied binary point is between bits 15 and 16. In an implementation with w implemented bits, the largest fraction of the cache that can be represented is 1- $(0.5)^{W}$.

Accessing MPAMCFG_CMAX

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_CMAX_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_CMAX_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_CMAX_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_CMAX_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_CMAX_s, MPAMCFG_CMAX_ns, MPAMCFG_CMAX_rt, and MPAMCFG_CMAX_rl must be separate registers:

- The Secure instance (MPAMCFG_CMAX_s) accesses the cache capacity partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_CMAX_ns) accesses the cache capacity partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_CMAX_rt) accesses the cache capacity partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_CMAX_rl) accesses the cache capacity partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the cache resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_CMAX can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0108	MPAMCFG_CMAX_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0108	MPAMCFG_CMAX_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0108	MPAMCFG_CMAX_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance

MPAM	MPAMF_BASE_rl	0x0108	MPAMCFG_CMAX_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_CMIN, MPAM Cache Minimum Capacity Partition Configuration Register

The MPAMCFG_CMIN characteristics are:

Purpose

The MPAMCFG_CMIN is a 32-bit read/write register that controls the fraction of the cache capacity that the PARTID selected by MPAMCFG_PART_SEL has priority to allocate.

MPAMCFG_CMIN_s controls the cache minimum capacity for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_CMIN_ns controls the cache minimum capacity for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. MPAMCFG_CMIN_rl controls the cache minimum capacity for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL. MPAMCFG_CMIN_rt controls the cache minimum capacity for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configuration

The power domain of MPAMCFG_CMIN is IMPLEMENTATION DEFINED.

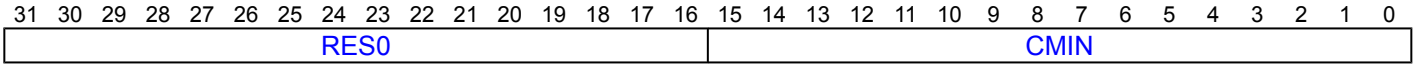
This register is present only when MPAMF_IDR.HAS_CCAP_PART == 1, (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented), and MPAMF_CCAP_IDR.HAS_CMIN == 1. Otherwise, direct accesses to MPAMCFG_CMIN are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_CMIN is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

CMIN, bits [15:0]

Minimum cache capacity usage in fixed-point fraction format by the partition selected by MPAMCFG_PART_SEL. The fraction represents the portion of the total cache capacity that the PARTID has priority to allocate.

The implemented width of the fixed-point fraction is the same as the width of MPAMCFG_CMAX.CMAX which is given in MPAMF_CCAP_IDR.CMAX_WD. Unimplemented bits within the field are RAZ/WI. The implemented bits of the CMIN field are always the most significant bits of the field.

The fixed-point fraction CMIN is less than 1. The implied binary point is between bits 15 and 16. In an implementation with w implemented bits, the largest fraction of the cache that can be represented is $1- (0.5)^{W}$.

Accessing MPAMCFG_CMIN

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_CMIN_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_CMIN_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_CMIN_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_CMIN_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_CMIN_s, MPAMCFG_CMIN_ns, MPAMCFG_CMIN_rt, and MPAMCFG_CMIN_rl must be separate registers:

- The Secure instance (MPAMCFG_CMIN_s) accesses the cache minimum capacity partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_CMIN_ns) accesses the cache minimum capacity partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_CMIN_rt) accesses the cache minimum capacity partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_CMIN_rl) accesses the cache minimum capacity partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_CMIN access the cache minimum capacity partitioning configuration settings for the cache resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_CMIN access the cache minimum capacity partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_CMIN access the cache minimum capacity partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_CMIN access the cache minimum capacity partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_CMIN can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0110	MPAMCFG_CMIN_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0110	MPAMCFG_CMIN_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0110	MPAMCFG_CMIN_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0110	MPAMCFG_CMIN_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_CPBM<n>, MPAM Cache Portion Bitmap Partition Configuration Register, n = 0 - 1023

The MPAMCFG_CPBM<n> characteristics are:

Purpose

The MPAMCFG_CPBM<n> register array gives access to the cache portion bitmap. Each register in the array is a read/write register that configures the cache portions numbered from <n * 32> to <31 + (n * 32)> that a PARTID is allowed to allocate.

After setting MPAMCFG_PART_SEL with a PARTID, software writes to the MPAMCFG_CPBM<n> register to configure which cache portions the PARTID is allowed to allocate.

The MPAMCFG_CPBM<n> register that contains the bitmap bit corresponding to cache portion p has n equal to p[15:5]. The field, P<x>, of that MPAMCFG_CPBM<n> register that contains the bitmap bit corresponding to cache portion p has x equal to p[4:0].

MPAMCFG_CPBM<n>_s controls cache portions for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL.
MPAMCFG_CPBM<n>_ns controls the cache portions for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.
MPAMCFG_CPBM<n>_rt controls cache portions for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL.
MPAMCFG_CPBM<n>_rl controls the cache portions for the Realm PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configuration

The power domain of MPAMCFG_CPBM<n> is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_CPOR_PART == 1. Otherwise, direct accesses to MPAMCFG_CPBM<n> are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_CPBM<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<x>, bit [x], for x = 31 to 0

Portion allocation control bit. Each cache portion allocation control bit, MPAMCFG_CPBM<n>.P<x>, grants permission to the PARTID selected by MPAMCFG_PART_SEL to allocate cache lines within cache portion <n*32> + x.

P<x>	Meaning
0b0	The PARTID is not permitted to allocate into cache portion <n * 32> + x.
0b1	The PARTID is permitted to allocate within cache portion <n * 32> + x.

The number of bits in the cache portion partitioning bit map of this component is given in MPAMF_CPOR_IDR.CPBM_WD. MPAMF_CPOR_IDR.CPBM_WD contains a value from 1 to 2^{15}, inclusive. Values of MPAMF_CPOR_IDR.CPBM_WD greater than 32 require an array of 32-bit MPAMCFG_CPBM<n> registers to access the cache portion bitmap, up to 1024 registers.

When (n * 32) + x > UInt(MPAMF_CPOR_IDR.CPBM_WD), access to this field is **RES0**.

Accessing MPAMCFG_CPBM<n>

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_CPBM<n>_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_CPBM<n>_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_CPBM<n>_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_CPBM<n>_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_CPBM<n>_s, MPAMCFG_CPBM<n>_ns, MPAMCFG_CPBM<n>_rt, and MPAMCFG_CPBM<n>_rl must be separate registers:

- The Secure instance (MPAMCFG_CPBM<n>_s) accesses the cache portion bitmap used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_CPBM<n>_ns) accesses the cache portion bitmap used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_CPBM<n>_rt) accesses the cache portion bitmap used for Root PARTIDs.
- The Realm instance (MPAMCFG_CPBM<n>_rl) accesses the cache portion bitmap used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the cache resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_CPBM<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x1000` + (4 * n)	MPAMCFG_CPBM<n>_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	`0x1000` + (4 * n)	MPAMCFG_CPBM<n>_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x1000` + (4 * n)	MPAMCFG_CPBM<n>_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x1000` + (4 * n)	MPAMCFG_CPBM<n>_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_DIS, MPAM Partition Configuration Disable Register

The MPAMCFG_DIS characteristics are:

Purpose

Disables a PARTID configuration as set in other MPAMCFG registers.

MPAMCFG_DIS_s disables a Secure PARTID. MPAMCFG_DIS_ns disables a Non-secure PARTID. MPAMCFG_DIS_rl disables a Realm PARTID. MPAMCFG_DIS_rt disables a Root PARTID.

Configuration

The power domain of MPAMCFG_DIS is IMPLEMENTATION DEFINED.

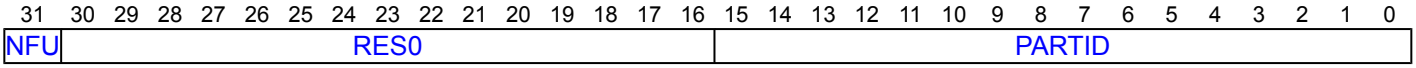
This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ENDIS == 1. Otherwise, direct accesses to MPAMCFG_DIS are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_DIS is a 32-bit register.

Field descriptions



NFU, bit [31]

When MPAMF_IDR.HAS_NFU == 1:

No Future Use. Software indicates that the PARTID disabled with NFU of 1 will not be used again and will be reconfigured and reenabled before being used again.

NFU	Meaning
0b0	Control settings of the disabled PARTID must be retained.
0b1	Control settings of the disabled PARTID may take an UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [30:16]

Reserved, RES0.

PARTID, bits [15:0]

Selects the PARTID to disable.

Accessing MPAMCFG_DIS

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_DIS_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_DIS_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_DIS_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_DIS_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_DIS_s, MPAMCFG_DIS_ns, MPAMCFG_DIS_rt, and MPAMCFG_DIS_rl must be separate registers:

- The Secure instance (MPAMCFG_DIS_s) accesses the PARTID disable used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_DIS_ns) accesses the PARTID disable used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_DIS_rt) accesses the PARTID disable used for Root PARTIDs.
- The Realm instance (MPAMCFG_DIS_rl) accesses the PARTID disable used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_DIS access the PARTID disable configuration settings for the PARTID disable resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_DIS access the PARTID disable configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_DIS access the PARTID disable configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_DIS access the PARTID disable configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_DIS can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0310	MPAMCFG_DIS_s

Accesses to this register are **WO/RAZ**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0310	MPAMCFG_DIS_ns

Accesses to this register are **WO/RAZ**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0310	MPAMCFG_DIS_rt

When FEAT_RME is implemented, accesses to this register are **WO/RAZ**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0310	MPAMCFG_DIS_rl

When FEAT_RME is implemented, accesses to this register are **WO/RAZ**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_EN, MPAM Partition Configuration Enable Register

The MPAMCFG_EN characteristics are:

Purpose

Enables a PARTID configuration as set in other MPAMCFG registers.

MPAMCFG_EN_s enables a Secure PARTID. MPAMCFG_EN_ns enables a Non-secure PARTID. MPAMCFG_EN_rl enables a Realm PARTID. MPAMCFG_EN_rt enables a Root PARTID.

Configuration

The power domain of MPAMCFG_EN is IMPLEMENTATION DEFINED.

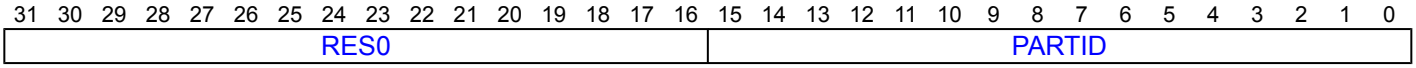
This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ENDIS == 1. Otherwise, direct accesses to MPAMCFG_EN are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_EN is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

PARTID, bits [15:0]

Selects the PARTID to enable.

Accessing MPAMCFG_EN

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_EN_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_EN_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_EN_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_EN_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_EN_s, MPAMCFG_EN_ns, MPAMCFG_EN_rt, and MPAMCFG_EN_rl must be separate registers:

- The Secure instance (MPAMCFG_EN_s) accesses the PARTID enable used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_EN_ns) accesses the PARTID enable used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_EN_rt) accesses the PARTID enable used for Root PARTIDs.
- The Realm instance (MPAMCFG_EN_rl) accesses the PARTID enable used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_EN access the PARTID enable configuration settings for the PARTID enable resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_EN access the PARTID enable configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_EN access the PARTID enable configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_EN access the PARTID enable configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_EN can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0300	MPAMCFG_EN_s

Accesses to this register are **WO/RAZ**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0300	MPAMCFG_EN_ns

Accesses to this register are **WO/RAZ**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0300	MPAMCFG_EN_rt

When FEAT_RME is implemented, accesses to this register are **WO/RAZ**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0300	MPAMCFG_EN_rl

When FEAT_RME is implemented, accesses to this register are **WO/RAZ**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_EN_FLAGS, MPAM Partition Configuration Enable Flags Register

The MPAMCFG_EN_FLAGS characteristics are:

Purpose

Enable flags for 32 PARTIDs.

MPAMCFG_EN_FLAGS_s gives read/write access to 32 Secure PARTIDs. MPAMCFG_EN_FLAGS_ns gives read/write access to 32 Non-secure PARTIDs. MPAMCFG_EN_FLAGS_rl gives read/write access to 32 Realm PARTIDs. MPAMCFG_EN_FLAGS_rt gives read/write access to 32 Root PARTIDs.

Configuration

The power domain of MPAMCFG_EN_FLAGS is IMPLEMENTATION DEFINED.

This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ENDIS == 1. Otherwise, direct accesses to MPAMCFG_EN_FLAGS are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_EN_FLAGS is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9
EN31	EN30	EN29	EN28	EN27	EN26	EN25	EN24	EN23	EN22	EN21	EN20	EN19	EN18	EN17	EN16	EN15	EN14	EN13	EN12	EN11	EN10	EN

EN<x>, bit [x], for x = 31 to 0

PARTID Enable flags. The group of flags accessed is selected by MPAMCFG_PART_SEL.PARTID_SEL & `0xFFE0` in bit [0] to (MPAMCFG_PART_SEL.PARTID_SEL & `0xFFE0`) + 31 in bit [31].

EN<x>	Meaning
`0b0`	The PARTID is disabled.
`0b1`	The PARTID is enabled.

Each bit in MPAMCFG_EN_FLAGS gives access to the same state as controlled by MPAMCFG_EN and MPAMCFG_DIS.

Bits MPAMCFG_EN_FLAGS.EN<x>, where (MPAMCFG_PART_SEL.PARTID_SEL & `0xFFE0`) + x is greater than MPAMF_IDR.PARTID_MAX, are not required to be implemented.

As with other partitioning controls, the enable flag for PARTID 0 must be reset to `0b1` (enabled).

Accessing MPAMCFG_EN_FLAGS

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_EN_FLAGS_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_EN_FLAGS_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_EN_FLAGS_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_EN_FLAGS_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_EN_FLAGS_s, MPAMCFG_EN_FLAGS_ns, MPAMCFG_EN_FLAGS_rt, and MPAMCFG_EN_FLAGS_rl must be separate registers:

- The Secure instance (MPAMCFG_EN_FLAGS_s) accesses the PARTID enable used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_EN_FLAGS_ns) accesses the PARTID enable used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_EN_FLAGS_rt) accesses the PARTID enable used for Root PARTIDs.
- The Realm instance (MPAMCFG_EN_FLAGS_rl) accesses the PARTID enable used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_EN_FLAGS access the PARTID enable configuration settings for the PARTID enable resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_EN_FLAGS access the PARTID enable configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_EN_FLAGS access the PARTID enable configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_EN_FLAGS access the PARTID enable configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_EN_FLAGS can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0320	MPAMCFG_EN_FLAGS_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0320	MPAMCFG_EN_FLAGS_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0320	MPAMCFG_EN_FLAGS_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0320	MPAMCFG_EN_FLAGS_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_IN_TL, MPAM Ingress PARTID Translation Configuration Register

The MPAMCFG_IN_TL characteristics are:

Purpose

Enables ingress PARTID translation and configures direct ingress PARTID translations.

Configuration

The power domain of MPAMCFG_IN_TL is IMPLEMENTATION DEFINED.

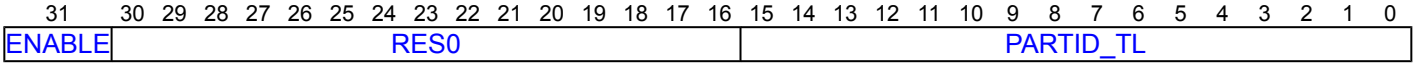
This register is present only when FEAT_MPAM_MSC_DOMAINS is implemented and MPAMF_IDR.HAS_IN_TL == 1. Otherwise, direct accesses to MPAMCFG_IN_TL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_IN_TL is a 32-bit register.

Field descriptions



ENABLE, bit [31]

Enables ingress PARTID translation in the MSC.

ENABLE	Meaning
0b0	Ingress PARTID translation is disabled.
0b1	Ingress PARTID translation is enabled.

Bits [30:16]

Reserved, RES0.

PARTID_TL, bits [15:0]

When MPAMF_IN_TL_IDR.HAS_DIRECT_TL == 1:

PARTID to be used as direct translation of the ingress PARTID configured in MPAMCFG_PART_SEL.PARTID_SEL when MPAMCFG_PART_SEL.INGRESS_TL == 1.

Otherwise:

Reserved, RES0.

Accessing MPAMCFG_IN_TL

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_IN_TL_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_IN_TL_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_IN_TL_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_IN_TL_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_IN_TL_s, MPAMCFG_IN_TL_ns, MPAMCFG_IN_TL_rt, and MPAMCFG_IN_TL_rl must be separate registers:

- The Secure instance (MPAMCFG_IN_TL_s) accesses the ingress PARTID translation used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_IN_TL_ns) accesses the ingress PARTID translation used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_IN_TL_rt) accesses the ingress PARTID translation used for Root PARTIDs.
- The Realm instance (MPAMCFG_IN_TL_rl) accesses the ingress PARTID translation used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_IN_TL access the ingress PARTID translation configuration settings without being affected by MPAMCFG_PART_SEL.RIS.

MPAMCFG_IN_TL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x3008	MPAMCFG_IN_TL_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x3008	MPAMCFG_IN_TL_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x3008	MPAMCFG_IN_TL_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x3008	MPAMCFG_IN_TL_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_IN_TL_BASE, MPAM Ingress PARTID Translation Base Configuration Register

The MPAMCFG_IN_TL_BASE characteristics are:

Purpose

Configures the base value used to compute translation PARTIDs for ingress PARTIDs that do not have direct translation.

Configuration

The power domain of MPAMCFG_IN_TL_BASE is IMPLEMENTATION DEFINED.

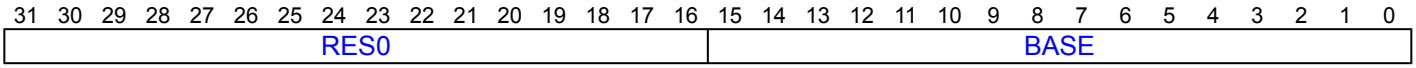
This register is present only when FEAT_MPAM_MSC_DOMAINS is implemented, MPAMF_IDR.HAS_IN_TL == 1, and MPAMF_IN_TL_IDR.HAS_BASE_MASK == 1. Otherwise, direct accesses to MPAMCFG_IN_TL_BASE are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_IN_TL_BASE is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

BASE, bits [15:0]

Base value used to compute the ingress translation of PARTIDs that do not have a direct translation configured.

Accessing MPAMCFG_IN_TL_BASE

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_IN_TL_BASE_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_IN_TL_BASE_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_IN_TL_BASE_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_IN_TL_BASE_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_IN_TL_BASE_s, MPAMCFG_IN_TL_BASE_ns, MPAMCFG_IN_TL_BASE_rt, and MPAMCFG_IN_TL_BASE_rl must be separate registers:

- The Secure instance (MPAMCFG_IN_TL_BASE_s) accesses the ingress PARTID translation base used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_IN_TL_BASE_ns) accesses the ingress PARTID translation base used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_IN_TL_BASE_rt) accesses the ingress PARTID translation base used for Root PARTIDs.
- The Realm instance (MPAMCFG_IN_TL_BASE_rl) accesses the ingress PARTID translation base used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_IN_TL_BASE access the ingress PARTID translation base configuration settings without being affected by MPAMCFG_PART_SEL.RIS.

MPAMCFG_IN_TL_BASE can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x3010	MPAMCFG_IN_TL_BASE_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x3010	MPAMCFG_IN_TL_BASE_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x3010	MPAMCFG_IN_TL_BASE_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x3010	MPAMCFG_IN_TL_BASE_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_IN_TL_MASK, MPAM Ingress PARTID Translation Mask Configuration Register

The MPAMCFG_IN_TL_MASK characteristics are:

Purpose

Configures the mask value used to compute translation PARTIDs for ingress PARTIDs that do not have direct translation.

Configuration

The power domain of MPAMCFG_IN_TL_MASK is IMPLEMENTATION DEFINED.

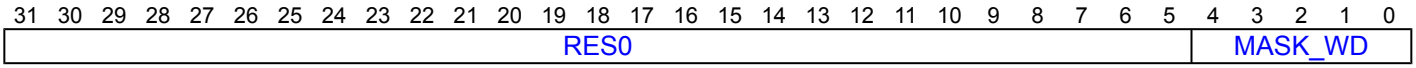
This register is present only when FEAT_MPAM_MSC_DOMAINS is implemented, MPAMF_IDR.HAS_IN_TL == 1, and MPAMF_IN_TL_IDR.HAS_BASE_MASK == 1. Otherwise, direct accesses to MPAMCFG_IN_TL_MASK are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_IN_TL_MASK is a 32-bit register.

Field descriptions



Bits [31:5]

Reserved, RES0.

MASK_WD, bits [4:0]

Value used to calculate the mask as 2^{MASK_WD}-1. The mask is then used to compute the ingress translation of PARTIDs that do not have a direct translation configured.

Accessing MPAMCFG_IN_TL_MASK

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_IN_TL_MASK_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_IN_TL_MASK_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_IN_TL_MASK_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_IN_TL_MASK_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_IN_TL_MASK_s, MPAMCFG_IN_TL_MASK_ns, MPAMCFG_IN_TL_MASK_rt, and MPAMCFG_IN_TL_MASK_rl must be separate registers:

- The Secure instance (MPAMCFG_IN_TL_MASK_s) accesses the ingress PARTID translation mask used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_IN_TL_MASK_ns) accesses the ingress PARTID translation mask used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_IN_TL_MASK_rt) accesses the ingress PARTID translation mask used for Root PARTIDs.
- The Realm instance (MPAMCFG_IN_TL_MASK_rl) accesses the ingress PARTID translation mask used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_IN_TL_MASK access the ingress PARTID translation mask configuration settings without being affected by MPAMCFG_PART_SEL.RIS.

MPAMCFG_IN_TL_MASK can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x3018	MPAMCFG_IN_TL_MASK_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x3018	MPAMCFG_IN_TL_MASK_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x3018	MPAMCFG_IN_TL_MASK_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x3018	MPAMCFG_IN_TL_MASK_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_INTPARTID, MPAM Internal PARTID Narrowing Configuration Register

The MPAMCFG_INTPARTID characteristics are:

Purpose

MPAMCFG_INTPARTID is a 32-bit read/write register that controls the mapping of the PARTID selected by MPAMCFG_PART_SEL into a narrower internal PARTID (intPARTID).

MPAMCFG_INTPARTID_s controls the mapping for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL.
MPAMCFG_INTPARTID_ns controls the mapping for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.
MPAMCFG_INTPARTID_rt controls the mapping for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL.
MPAMCFG_INTPARTID_rl controls the mapping for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

The MPAMCFG_INTPARTID register associates the request PARTID (reqPARTID) in the MPAMCFG_PART_SEL register with an internal PARTID (intPARTID) in this register. To set that association, store reqPARTID into the MPAMCFG_PART_SEL register and then store the intPARTID into the MPAMCFG_INTPARTID register. To read the association, store reqPARTID into the MPAMCFG_PART_SEL register and then read MPAMCFG_INTPARTID.

If the intPARTID stored into MPAMCFG_INTPARTID is out-of-range or does not have the INTERNAL bit set, the association of reqPARTID to intPARTID is not written and MPAMF_ESR is set to indicate an intPARTID_Range error.

If MPAMCFG_PART_SEL.INTERNAL is 1 when MPAMCFG_INTPARTID is read or written, MPAMF_ESR is set to indicate an Unexpected_INTERNAL error.

Configuration

The power domain of MPAMCFG_INTPARTID is IMPLEMENTATION DEFINED.

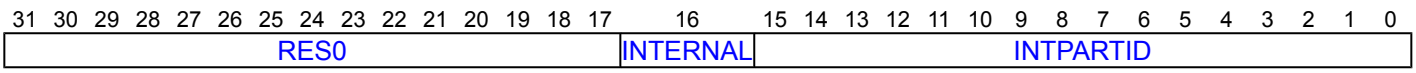
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PARTID_NRW == 1. Otherwise, direct accesses to MPAMCFG_INTPARTID are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_INTPARTID is a 32-bit register.

Field descriptions



Bits [31:17]

Reserved, RES0.

INTERNAL, bit [16]

Internal PARTID flag.

This bit must be 1 when written to the register. If written as 0, the write will not update the reqPARTID to intPARTID association.

On a read of this register, the bit will always read the value last written.

INTPARTID, bits [15:0]

This field contains the intPARTID mapped to the reqPARTID in MPAMCFG_PART_SEL.

The maximum intPARTID supported is MPAMF_PARTID_NRW_IDR.INTPARTID_MAX.

Accessing MPAMCFG_INTPARTID

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_INTPARTID_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_INTPARTID_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_INTPARTID_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_INTPARTID_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_INTPARTID_s, MPAMCFG_INTPARTID_ns, MPAMCFG_INTPARTID_rt, and MPAMCFG_INTPARTID_rl must be separate registers:

- The Secure instance (MPAMCFG_INTPARTID_s) accesses the PARTID narrowing used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_INTPARTID_ns) accesses the PARTID narrowing used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_INTPARTID_rt) accesses the PARTID narrowing used for Root PARTIDs.
- The Realm instance (MPAMCFG_INTPARTID_rl) accesses the PARTID narrowing used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_INTPARTID access the PARTID narrowing configuration settings without being affected by MPAMCFG_PART_SEL.RIS.

Loads and stores to MPAMCFG_INTPARTID access the PARTID narrowing configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_INTPARTID can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0600	MPAMCFG_INTPARTID_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0600	MPAMCFG_INTPARTID_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0600	MPAMCFG_INTPARTID_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0600	MPAMCFG_INTPARTID_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_MBW_MAX, MPAM Memory Bandwidth Maximum Partition Configuration Register

The MPAMCFG_MBW_MAX characteristics are:

Purpose

MPAMCFG_MBW_MAX is a 32-bit read/write register that controls the maximum fraction of memory bandwidth that the PARTID selected by MPAMCFG_PART_SEL is permitted to use.

MPAMCFG_MBW_MAX_s controls maximum bandwidth for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MAX_ns controls the maximum bandwidth for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MAX_rt controls the maximum bandwidth for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MAX_rl controls the maximum bandwidth for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

A PARTID that has used more than MAX is given no access to additional bandwidth if HARDLIM == 1 or is given additional bandwidth only if there are no requests from PARTIDs that have not exceeded their MAX if HARDLIM == 0.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configuration

The power domain of MPAMCFG_MBW_MAX is IMPLEMENTATION DEFINED.

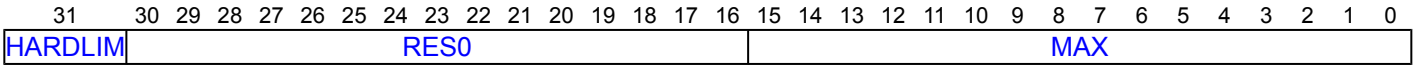
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1, and MPAMF_MBW_IDR.HAS_MAX == 1. Otherwise, direct accesses to MPAMCFG_MBW_MAX are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_MAX is a 32-bit register.

Field descriptions



HARDLIM, bit [31]

Maximum-bandwidth limit behaviour selection.

HARDLIM	Meaning
0b0	Soft limit: when MAX bandwidth is exceeded, the partition contends with a low preference for downstream bandwidth beyond MAX.
0b1	Hard limit: when MAX bandwidth is exceeded, the partition does not use any more bandwidth until the memory bandwidth measurement for the partition falls below MAX.

Accessing this field has the following behavior:

- When MPAMF_MBW_IDR.MAX_LIM == 0b00, access to this field is **RW**.
- When MPAMF_MBW_IDR.MAX_LIM == 0b01, access to this field is **RAZ/WI**.
- When MPAMF_MBW_IDR.MAX_LIM == 0b10, access to this field is **RAO/WI**.

Bits [30:16]

Reserved, RES0.

MAX, bits [15:0]

Memory maximum bandwidth allocated to the partition selected by MPAMCFG_PART_SEL. MAX is in fixed-point fraction format. The fraction represents the portion of the total memory bandwidth capacity through the controlled component that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in MPAMF_MBW_IDR.BWA_WD. Unimplemented bits are RAZ/WI. The implemented bits of the MAX field are always to the left of the field. For example, if BWA_WD = 3, the implemented bits are MPAMCFG_MBW_MAX[15:13] and MPAMCFG_MBW_MAX[12:0] are unimplemented.

The fixed-point fraction MAX is less than 1. The implied binary point is between bits 15 and 16. In an implementation with w implemented bits, the largest fraction of the bandwidth that can be represented is 1- $(0.5)^{w}$.

Accessing MPAMCFG_MBW_MAX

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_MBW_MAX_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_MAX_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_MAX_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_MAX_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_MAX_s, MPAMCFG_MBW_MAX_ns, MPAMCFG_MBW_MAX_rt, and MPAMCFG_MBW_MAX_rl must be separate registers:

- The Secure instance (MPAMCFG_MBW_MAX_s) accesses the memory maximum bandwidth partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_MAX_ns) accesses the memory maximum bandwidth partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_MAX_rt) accesses the memory maximum bandwidth partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_MAX_rl) accesses the memory maximum bandwidth partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_MBW_MAX can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0208	MPAMCFG_MBW_MAX_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0208	MPAMCFG_MBW_MAX_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0208	MPAMCFG_MBW_MAX_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0208	MPAMCFG_MBW_MAX_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_MBW_MIN, MPAM Memory Bandwidth Minimum Partition Configuration Register

The MPAMCFG_MBW_MIN characteristics are:

Purpose

MPAMCFG_MBW_MIN is a 32-bit read/write register that controls the minimum fraction of memory bandwidth that the PARTID selected by MPAMCFG_PART_SEL is permitted to use.

MPAMCFG_MBW_MIN_s controls the minimum bandwidth for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MIN_ns controls the minimum bandwidth for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MIN_rt controls the minimum bandwidth for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MIN_rl controls the minimum bandwidth for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

A PARTID that has used less than MIN is given preferential access to bandwidth.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configuration

The power domain of MPAMCFG_MBW_MIN is IMPLEMENTATION DEFINED.

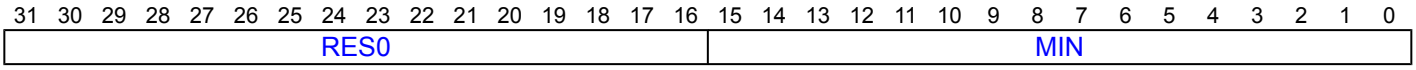
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1, and MPAMF_MBW_IDR.HAS_MIN == 1. Otherwise, direct accesses to MPAMCFG_MBW_MIN are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_MIN is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

MIN, bits [15:0]

Memory minimum bandwidth allocated to the partition selected by MPAMCFG_PART_SEL. MIN is in fixed-point fraction format. The fraction represents the portion of the total memory bandwidth capacity through the controlled component that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in MPAMF_MBW_IDR.BWA_WD. Unimplemented bits are RAZ/WI. The implemented bits of the MIN field are always to the left of the field. For example, if BWA_WD = 4, the implemented bits are MPAMCFG_MBW_MIN[15:12] and MPAMCFG_MBW_MIN[11:0] are unimplemented.

The fixed-point fraction MIN is less than 1. The implied binary point is between bits 15 and 16. In an implementation with w implemented bits, the largest fraction of the bandwidth that can be represented is $1- (0.5)^{W}$.

Accessing MPAMCFG_MBW_MIN

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_MBW_MIN_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_MIN_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_MIN_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_MIN_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_MIN_s, MPAMCFG_MBW_MIN_ns, MPAMCFG_MBW_MIN_rt, and MPAMCFG_MBW_MIN_rl must be separate registers:

- The Secure instance (MPAMCFG_MBW_MIN_s) accesses the memory minimum bandwidth partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_MIN_ns) accesses the memory minimum bandwidth partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_MIN_rt) accesses the memory minimum bandwidth partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_MIN_rl) accesses the memory minimum bandwidth partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_MBW_MIN can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0200	MPAMCFG_MBW_MIN_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0200	MPAMCFG_MBW_MIN_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0200	MPAMCFG_MBW_MIN_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0200	MPAMCFG_MBW_MIN_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_MBW_PBM<n>, MPAM Bandwidth Portion Bitmap Partition Configuration Register, n = 0 - 127

The MPAMCFG_MBW_PBM<n> characteristics are:

Purpose

The MPAMCFG_MBW_PBM<n> register array gives access to the memory bandwidth portion bitmap. Each register in the array is a read/write register that configures whether a PARTID is allowed to allocate bandwidth portions within a range.

The range of portions covered in MPAMCFG_MBW_PBM<n> is from portion <32*n> to portion <32*n +31>.

After setting MPAMCFG_PART_SEL with a PARTID, software writes to one or more of the MPAMCFG_MBW_PBM<n> registers to configure with bandwidth portions the PARTID is allowed to allocate.

The MPAMCFG_MBW_PBM<n> register that contains the bitmap bit corresponding to memory bandwidth portion p has n equal to p[11:5]. The field, P<x> of that MPAMCFG_MBW_PBM<n> register that contains the bitmap bit corresponding to memory bandwidth portion p has <x> equal to p[4:0].

The MPAMCFG_MBW_PBM<n>_s registers control the bandwidth portion bitmap for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. The MPAMCFG_MBW_PBM<n>_ns registers control the bandwidth portion bitmap for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. The MPAMCFG_MBW_PBM<n>_rt registers control the bandwidth portion bitmap for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. The MPAMCFG_MBW_PBM<n>_rl registers control the bandwidth portion bitmap for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configuration

The power domain of MPAMCFG_MBW_PBM<n> is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1, and MPAMF_MBW_IDR.HAS_PBM == 1. Otherwise, direct accesses to MPAMCFG_MBW_PBM<n> are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_PBM<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<x>, bit [x], for x = 31 to 0

Portion allocation control bit. Each bandwidth portion allocation control bit MPAMCFG_MBW_PBM<n>.P<x> grants permission to the PARTID selected by MPAMCFG_PART_SEL to allocate bandwidth within bandwidth portion <32*n> + <x>.

P<x>	Meaning
0b0	The PARTID is not permitted to allocate into bandwidth portion <32*n> + <x>.
0b1	The PARTID is permitted to allocate within bandwidth portion <32*n> + <x>.

The number of bits in the bandwidth portion partitioning bit map of this component is given in MPAMF_MBW_IDR.BWPBM_WD. BWPBM_WD contains a value from 1 to 2^{12}, inclusive. Values of MPAMF_MBW_IDR.BWPBM_WD greater than 32 require a group of 32-bit registers to access the bandwidth portion bitmap, up to 128 32-bit registers.

When (n * 32) + x > UInt(MPAMF_MBW_IDR.BWPBM_WD), access to this field is **RES0**.

Accessing MPAMCFG_MBW_PBM<n>

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_MBW_PBM<n>_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_PBM<n>_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_PBM<n>_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_PBM<n>_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_PBM<n>_s, MPAMCFG_MBW_PBM<n>_ns, MPAMCFG_MBW_PBM<n>_rt, and MPAMCFG_MBW_PBM<n>_rl must be separate registers:

- The Secure instance (MPAMCFG_MBW_PBM<n>_s) accesses the memory bandwidth portion bitmap used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_PBM<n>_ns) accesses the memory bandwidth portion bitmap used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_PBM<n>_rt) accesses the memory bandwidth portion bitmap used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_PBM<n>_rl) accesses the memory bandwidth portion bitmap used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_MBW_PBM<n> can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x2000 + (4 * n)	MPAMCFG_MBW_PBM<n>_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_MBW_PROP, MPAM Memory Bandwidth Proportional Stride Partition Configuration Register

The MPAMCFG_MBW_PROP characteristics are:

Purpose

Controls the proportional stride of memory bandwidth that the PARTID selected by MPAMCFG_PART_SEL uses.

MPAMCFG_MBW_PROP_s controls the bandwidth proportional stride for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_PROP_ns controls the bandwidth proportional stride for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_PROP_rt controls the bandwidth proportional stride for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_PROP_rl controls the bandwidth proportional stride for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

Proportional stride is a relative cost of bandwidth requested by one PARTID in relation to the costs of the bandwidths requested by each other PARTID also competing to use the bandwidth.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configuration

The power domain of MPAMCFG_MBW_PROP is IMPLEMENTATION DEFINED.

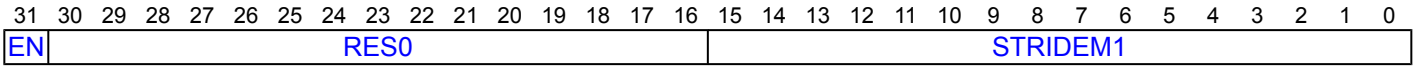
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1, and MPAMF_MBW_IDR.HAS_PROP == 1. Otherwise, direct accesses to MPAMCFG_MBW_PROP are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_PROP is a 32-bit register.

Field descriptions



EN, bit [31]

Enable proportional stride bandwidth partitioning.

EN	Meaning
0b0	The selected partition is not regulated by proportional stride bandwidth partitioning.
0b1	The selected partition has bandwidth usage regulated by proportional stride bandwidth partitioning as controlled by STRIDEM1.

Bits [30:16]

Reserved, RES0.

STRIDEM1, bits [15:0]

Memory bandwidth stride minus 1 allocated to the partition selected by MPAMCFG_PART_SEL. STRIDEM1 represents the normalized cost of bandwidth consumption by the partition.

The proportional stride partitioning control parameter is an unsigned integer representing the normalized cost to a partition for consuming bandwidth. Larger values have a larger cost and correspond to a lesser allocation of bandwidth while smaller values indicate a lesser cost and therefore a higher allocation of bandwidth.

The implemented width of STRIDEM1 is given in MPAMF_MBW_IDR.BWA_WD.

Accessing MPAMCFG_MBW_PROP

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_MBW_PROP_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_PROP_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_PROP_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_PROP_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_PROP_s, MPAMCFG_MBW_PROP_ns, MPAMCFG_MBW_PROP_rt, and MPAMCFG_MBW_PROP_rl must be separate registers:

- The Secure instance (MPAMCFG_MBW_PROP_s) accesses the memory proportional stride bandwidth partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_PROP_ns) accesses the memory proportional stride bandwidth partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_PROP_rt) accesses the memory proportional stride bandwidth partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_PROP_rl) accesses the memory proportional stride bandwidth partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_MBW_PROP can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0500	MPAMCFG_MBW_PROP_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0500	MPAMCFG_MBW_PROP_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0500	MPAMCFG_MBW_PROP_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0500	MPAMCFG_MBW_PROP_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_MBW_WINWD, MPAM Memory Bandwidth Partitioning Window Width Configuration Register

The MPAMCFG_MBW_WINWD characteristics are:

Purpose

MPAMCFG_MBW_WINWD is a 32-bit register that shows and sets the value of the window width for the PARTID in MPAMCFG_PART_SEL.

MPAMCFG_MBW_WINWD_s reads and controls the bandwidth control window width for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_WINWD_ns reads and controls the bandwidth control window width for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_WINWD_rt reads and controls the bandwidth control window width for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_WINWD_rl reads and controls the bandwidth control window width for the Real PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

MPAMCFG_MBW_WINWD is read-only if MPAMF_MBW_IDR.WINDWR == 0, and the window width is set by the hardware, even if variable.

MPAMCFG_MBW_WINWD is read/write if MPAMF_MBW_IDR.WINDWR == 1, permitting configuration of the window width for each PARTID independently on hardware that supports this functionality.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configuration

The power domain of MPAMCFG_MBW_WINWD is IMPLEMENTATION DEFINED.

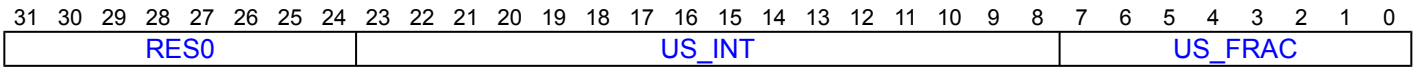
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_MBW_PART == 1. Otherwise, direct accesses to MPAMCFG_MBW_WINWD are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_WINWD is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

US_INT, bits [23:8]

Window width, integer microseconds.

This field reads (and sets) the integer part of the window width in microseconds for the PARTID selected by MPAMCFG_PART_SEL.

US_FRAC, bits [7:0]

Window width, fractional microseconds.

This field reads (and sets) the fractional part of the window width in microseconds for the PARTID selected by MPAMCFG_PART_SEL.

Accessing MPAMCFG_MBW_WINWD

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_MBW_WINWD_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_WINWD_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_WINWD_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_WINWD_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_WINWD_s, MPAMCFG_MBW_WINWD_ns, MPAMCFG_MBW_WINWD_rt, and MPAMCFG_MBW_WINWD_rl must be separate registers:

- The Secure instance (MPAMCFG_MBW_WINWD_s) accesses the window width used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_WINWD_ns) accesses the window width used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_WINWD_rt) accesses the window width used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_WINWD_rl) accesses the window width used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_MBW_WINWD can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0220	MPAMCFG_MBW_WINWD_s

Accessible as follows:

- When MPAMF_MBW_IDR.WINDWR == 0, accesses to this register are **RO**.
- When MPAMF_MBW_IDR.WINDWR == 1, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0220	MPAMCFG_MBW_WINWD_ns

Accessible as follows:

- When MPAMF_MBW_IDR.WINDWR == 0, accesses to this register are **RO**.
- When MPAMF_MBW_IDR.WINDWR == 1, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0220	MPAMCFG_MBW_WINWD_rt

Accessible as follows:

- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 0, accesses to this register are **RO**.
- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 1, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0220	MPAMCFG_MBW_WINWD_rl

Accessible as follows:

- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 0, accesses to this register are **RO**.
- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 1, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_OUT_TL, MPAM Egress PARTID Translation Configuration Register

The MPAMCFG_OUT_TL characteristics are:

Purpose

Enables egress PARTID translation and configures direct egress PARTID translations.

Configuration

The power domain of MPAMCFG_OUT_TL is IMPLEMENTATION DEFINED.

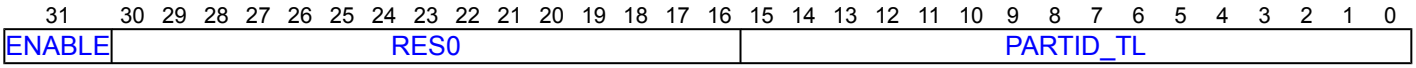
This register is present only when FEAT_MPAM_MSC_DOMAINS is implemented and MPAMF_IDR.HAS_OUT_TL == 1. Otherwise, direct accesses to MPAMCFG_OUT_TL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_OUT_TL is a 32-bit register.

Field descriptions



ENABLE, bit [31]

Enables egress PARTID translation in the MSC.

ENABLE	Meaning
0b0	Egress PARTID translation is disabled.
0b1	Egress PARTID translation is enabled.

Bits [30:16]

Reserved, RES0.

PARTID_TL, bits [15:0]

When MPAMF_OUT_TL_IDR.HAS_DIRECT_TL == 1:

PARTID to be used as direct translation of the egress PARTID configured in MPAMCFG_PART_SEL.PARTID_SEL when MPAMCFG_PART_SEL.INGRESS_TL == 0.

Otherwise:

Reserved, RES0.

Accessing MPAMCFG_OUT_TL

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_OUT_TL_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_OUT_TL_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_OUT_TL_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_OUT_TL_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_OUT_TL_s, MPAMCFG_OUT_TL_ns, MPAMCFG_OUT_TL_rt, and MPAMCFG_OUT_TL_rl must be separate registers:

- The Secure instance (MPAMCFG_OUT_TL_s) accesses the egress PARTID translation used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_OUT_TL_ns) accesses the egress PARTID translation used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_OUT_TL_rt) accesses the egress PARTID translation used for Root PARTIDs.
- The Realm instance (MPAMCFG_OUT_TL_rl) accesses the egress PARTID translation used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_OUT_TL access the egress PARTID translation configuration settings without being affected by MPAMCFG_PART_SEL.RIS.

MPAMCFG_OUT_TL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x3208	MPAMCFG_OUT_TL_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x3208	MPAMCFG_OUT_TL_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x3208	MPAMCFG_OUT_TL_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x3208	MPAMCFG_OUT_TL_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_OUT_TL_BASE, MPAM Egress PARTID Translation Base Configuration Register

The MPAMCFG_OUT_TL_BASE characteristics are:

Purpose

Configures the base value used to compute translation PARTIDs for egress PARTIDs that do not have direct translation.

Configuration

The power domain of MPAMCFG_OUT_TL_BASE is IMPLEMENTATION DEFINED.

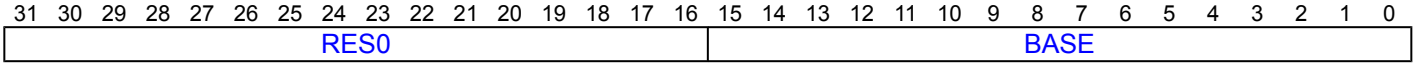
This register is present only when FEAT_MPAM_MSC_DOMAINS is implemented, MPAMF_IDR.HAS_OUT_TL == 1, and MPAMF_OUT_TL_IDR.HAS_BASE_MASK == 1. Otherwise, direct accesses to MPAMCFG_OUT_TL_BASE are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_OUT_TL_BASE is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

BASE, bits [15:0]

Base value used to compute the egress translation of PARTIDs that do not have a direct translation configured.

Accessing MPAMCFG_OUT_TL_BASE

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_OUT_TL_BASE_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_OUT_TL_BASE_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_OUT_TL_BASE_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_OUT_TL_BASE_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_OUT_TL_BASE_s, MPAMCFG_OUT_TL_BASE_ns, MPAMCFG_OUT_TL_BASE_rt, and MPAMCFG_OUT_TL_BASE_rl must be separate registers:

- The Secure instance (MPAMCFG_OUT_TL_BASE_s) accesses the egress PARTID translation base used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_OUT_TL_BASE_ns) accesses the egress PARTID translation base used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_OUT_TL_BASE_rt) accesses the egress PARTID translation base used for Root PARTIDs.
- The Realm instance (MPAMCFG_OUT_TL_BASE_rl) accesses the egress PARTID translation base used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_OUT_TL_BASE access the egress PARTID translation base configuration settings without being affected by MPAMCFG_PART_SEL.RIS.

MPAMCFG_OUT_TL_BASE can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x3210	MPAMCFG_OUT_TL_BASE_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x3210	MPAMCFG_OUT_TL_BASE_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x3210	MPAMCFG_OUT_TL_BASE_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x3210	MPAMCFG_OUT_TL_BASE_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_OUT_TL_MASK, MPAM Egress PARTID Translation Mask Configuration Register

The MPAMCFG_OUT_TL_MASK characteristics are:

Purpose

Configures the mask value used to compute translation PARTIDs for egress PARTIDs that do not have direct translation.

Configuration

The power domain of MPAMCFG_OUT_TL_MASK is IMPLEMENTATION DEFINED.

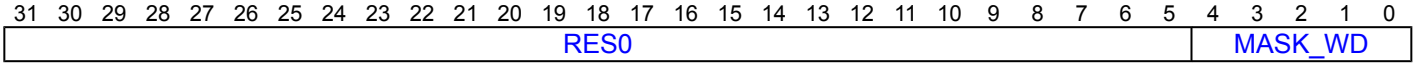
This register is present only when FEAT_MPAM_MSC_DOMAINS is implemented, MPAMF_IDR.HAS_OUT_TL == 1, and MPAMF_OUT_TL_IDR.HAS_BASE_MASK == 1. Otherwise, direct accesses to MPAMCFG_OUT_TL_MASK are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_OUT_TL_MASK is a 32-bit register.

Field descriptions



Bits [31:5]

Reserved, RES0.

MASK_WD, bits [4:0]

Value used to calculate the mask as 2^{MASK_WD}-1. The mask is then used to compute the egress translation of PARTIDs that do not have a direct translation configured.

Accessing MPAMCFG_OUT_TL_MASK

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_OUT_TL_MASK_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_OUT_TL_MASK_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_OUT_TL_MASK_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_OUT_TL_MASK_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_OUT_TL_MASK_s, MPAMCFG_OUT_TL_MASK_ns, MPAMCFG_OUT_TL_MASK_rt, and MPAMCFG_OUT_TL_MASK_rl must be separate registers:

- The Secure instance (MPAMCFG_OUT_TL_MASK_s) accesses the egress PARTID translation mask used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_OUT_TL_MASK_ns) accesses the egress PARTID translation mask used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_OUT_TL_MASK_rt) accesses the egress PARTID translation mask used for Root PARTIDs.
- The Realm instance (MPAMCFG_OUT_TL_MASK_rl) accesses the egress PARTID translation mask used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_OUT_TL_MASK access the egress PARTID translation mask configuration settings without being affected by MPAMCFG_PART_SEL.RIS.

MPAMCFG_OUT_TL_MASK can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x3218	MPAMCFG_OUT_TL_MASK_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x3218	MPAMCFG_OUT_TL_MASK_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x3218	MPAMCFG_OUT_TL_MASK_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x3218	MPAMCFG_OUT_TL_MASK_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_PART_SEL, MPAM Partition Configuration Selection Register

The MPAMCFG_PART_SEL characteristics are:

Purpose

Selects a partition ID to configure.

MPAMCFG_PART_SEL_s selects a Secure PARTID to configure. MPAMCFG_PART_SEL_ns selects a Non-secure PARTID to configure. MPAMCFG_PART_SEL_rt selects a Root PARTID to configure. MPAMCFG_PART_SEL_rl selects a Realm PARTID to configure.

After setting this register with a PARTID, software (usually a hypervisor) can perform a series of accesses to MPAMCFG registers to configure parameters for MPAM resource controls to use when requests have that PARTID.

Configuration

The power domain of MPAMCFG_PART_SEL is IMPLEMENTATION DEFINED.

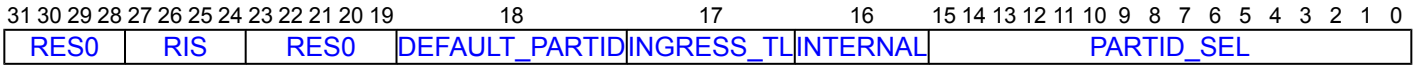
This register is present only when FEAT_MPAM is implemented and (MPAMF_IDR.HAS_CCAP_PART == 1, or MPAMF_IDR.HAS_CPOR_PART == 1, or MPAMF_IDR.HAS_MBW_PART == 1, or MPAMF_IDR.HAS_PRI_PART == 1, or MPAMF_IDR.HAS_PARTID_NRW == 1, or (MPAMF_IDR.EXT == 0 and MPAMF_IDR.HAS_IMPL_IDR == 1), or (MPAMF_IDR.EXT == 1, MPAMF_IDR.HAS_IMPL_IDR == 1, and MPAMF_IDR.NO_IMPL_PART == 0)). Otherwise, direct accesses to MPAMCFG_PART_SEL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_PART_SEL is a 32-bit register.

Field descriptions



Bits [31:28]

Reserved, RES0.

RIS, bits [27:24]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented), MPAMF_IDR.EXT == 1, and MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. RIS selects one resource to configure through MPAMCFG registers and describe with MPAMF ID registers.

Otherwise:

Reserved, RES0.

Bits [23:19]

Reserved, RES0.

DEFAULT_PARTID, bit [18]

When FEAT_MPAM_MSC_DCTRL is implemented:

Disables PARTID selection and instead configures the default resource control behavior.

DEFAULT_PARTID	Meaning
0b0	PARTID_SEL is interpreted as a request PARTID.
0b1	PARTID_SEL is ignored and any access to MPAMCFG registers is intended to be for configuring the default resource control behaviour.

Otherwise:

Reserved, RES0.

INGRESS_TL, bit [17]

When FEAT_MPAM_MSC_DOMAINS is implemented:

Indicates that PARTID_SEL is used for ingress translation.

INGRESS_TL	Meaning
0b0	PARTID_SEL is interpreted as a request PARTID for configuring MSC controls.
0b1	PARTID_SEL is interpreted as an untranslated ingress request PARTID to be used for configuring direct translations.

When MPAMF_IDR.HAS_IN_TL == 0, access to this field is **RAZ/WI**.

Otherwise:

Reserved, RES0.

INTERNAL, bit [16]

Internal PARTID.

If MPAMF_IDR.HAS_PARTID_NRW == 0b1:

INTERNAL	Meaning
0b0	PARTID_SEL is interpreted as a request PARTID and ignored except for use with MPAMCFG_INTPARTID register access.
0b1	PARTID_SEL is interpreted as an internal PARTID and used for access to MPAMCFG control settings except for MPAMCFG_INTPARTID.

If PARTID narrowing is implemented as indicated by MPAMF_IDR.HAS_PARTID_NRW = 1, when accessing other MPAMCFG registers the value of the MPAMCFG_PART_SEL.INTERNAL bit is checked for these conditions:

- When the MPAMCFG_INTPARTID register is read or written, if the value of MPAMCFG_PART_SEL.INTERNAL is not 0, an Unexpected_INTERNAL error is set in MPAMF_ESR.
- When an MPAMCFG register other than MPAMCFG_INTPARTID is read or written, if the value of MPAMCFG_PART_SEL.INTERNAL is not 1, MPAMF_ESR is set to indicate an intPARTID_Range error.

In either error case listed here, the value returned by a read operation is UNPREDICTABLE, and the control settings are not affected by a write.

When MPAMF_IDR.HAS_PARTID_NRW == 0, access to this field is **RAZ/WI**.

PARTID_SEL, bits [15:0]

Selects the partition ID to configure.

Reads and writes to other MPAMCFG registers are indexed by PARTID_SEL and by the NS bit used to access MPAMCFG_PART_SEL to access the configuration for a single partition.

Accessing MPAMCFG_PART_SEL

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_PART_SEL_s must only be accessible from the Secure MPAM feature page. MPAMCFG_PART_SEL_ns must only be accessible from the Non-secure MPAM feature page.

MPAMCFG_PART_SEL_s and MPAMCFG_PART_SEL_ns must be separate registers. The Secure instance (MPAMCFG_PART_SEL_s) accesses the PARTID selector used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_PART_SEL_ns) accesses the PARTID selector used for Non-secure PARTIDs.

MPAMCFG_PART_SEL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0100	MPAMCFG_PART_SEL_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0100	MPAMCFG_PART_SEL_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0100	MPAMCFG_PART_SEL_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0100	MPAMCFG_PART_SEL_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_PRI, MPAM Priority Partition Configuration Register

The MPAMCFG_PRI characteristics are:

Purpose

Controls the internal and downstream priority of requests attributed to the PARTID selected by MPAMCFG_PART_SEL.

MPAMCFG_PRI_s controls the priorities for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_PRI_ns controls the priorities for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. MPAMCFG_PRI_rt controls the priorities for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_PRI_rl controls the priorities for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configuration

The power domain of MPAMCFG_PRI is IMPLEMENTATION DEFINED.

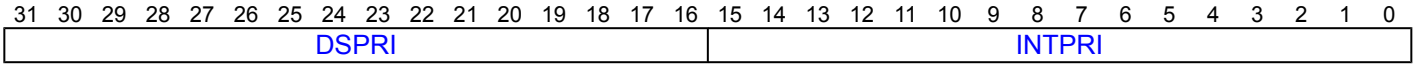
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PRI_PART == 1. Otherwise, direct accesses to MPAMCFG_PRI are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_PRI is a 32-bit register.

Field descriptions



DSPRI, bits [31:16]

Downstream priority.

If MPAMF_PRI_IDR.HAS_DSPRI == 0, bits of this field are RES0 as this field is not used.

If MPAMF_PRI_IDR.HAS_DSPRI == 1, this field is a priority value applied to downstream communications from this MSC for transactions of the partition selected by MPAMCFG_PART_SEL.

The implemented width of this field is MPAMF_PRI_IDR.DSPRI_WD bits. If the implemented width is less than the width of this field, the least significant bits are used.

The encoding of priority is 0-as-lowest or 0-as-highest priority according to the value of MPAMF_PRI_IDR.DSPRI_0_IS_LOW.

INTPRI, bits [15:0]

Internal priority.

If MPAMF_PRI_IDR.HAS_INTPRI == 0, bits of this field are RES0 as this field is not used.

If MPAMF_PRI_IDR.HAS_INTPRI == 1, this field is a priority value applied internally inside this MSC for transactions of the partition selected by MPAMCFG_PART_SEL.

The implemented width of this field is MPAMF_PRI_IDR.INTPRI_WD bits. If the implemented width is less than the width of this field, the least significant bits are used.

The encoding of priority is 0-as-lowest or 0-as-highest priority according to the value of MPAMF_PRI_IDR.INTPRI_0_IS_LOW.

Accessing MPAMCFG_PRI

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_PRI_s must only be accessible from the Secure MPAM feature page.
- MPAMCFG_PRI_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMCFG_PRI_rt must only be accessible from the Root MPAM feature page.
- MPAMCFG_PRI_rl must only be accessible from the Realm MPAM feature page.

MPAMCFG_PRI_s, MPAMCFG_PRI_ns, MPAMCFG_PRI_rt, and MPAMCFG_PRI_rl must be separate registers:

- The Secure instance (MPAMCFG_PRI_s) accesses the priority partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_PRI_ns) accesses the priority partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_PRI_rt) accesses the priority partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_PRI_rl) accesses the priority partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the priority resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_PRI can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0400	MPAMCFG_PRI_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0400	MPAMCFG_PRI_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0400	MPAMCFG_PRI_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0400	MPAMCFG_PRI_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_AIDR, MPAM Architecture Identification Register

The MPAMF_AIDR characteristics are:

Purpose

Identifies the version of the MPAM architecture that this MSC implements.

Note

The following values are defined for bits [7:0]: 0x01 == MPAM architecture v0.1 `0x10` == MPAM architecture v1.0 * `0x11` == MPAM architecture v1.1

Configuration

The power domain of MPAMF_AIDR is IMPLEMENTATION DEFINED.

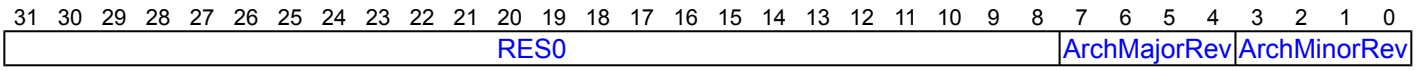
This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_AIDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_AIDR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

ArchMajorRev, bits [7:4]

Major revision of the MPAM architecture implemented by the MSC.

This table shows the only valid combinations of MPAM version numbers in an MSC. FORCE_NS functionality is only available in MPAM v0.1.

ArchMajorRev	ArchMinorRev	MPAMv	Available
0	0		None.
0	1	v0.1	MPAMv1.0 + MPAMv1.1 + FORCE_NS
1	0	v1.0	MPAMv1.0
1	1	v1.1	MPAMv1.0 + MPAMv1.1 - FORCE_NS

Use of MPAMv0.1 in MSCs is restricted to limited circumstances. The MSC must be able to initiate requests in the Secure address space which have MPAM PARTID forced to the Non-secure space with that forcing not controllable or observable by the software that configures the device for Secure requests. Please contact Arm before setting MPAMF_AIDR to report MPAMv0.1.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

ArchMinorRev, bits [3:0]

Minor revision of the MPAM architecture implemented by the MSC.

See the table in the description of the ArchMajorRev field in this register.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPAMF_AIDR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_AIDR is read-only.

MPAMF_AIDR must be readable from the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_AIDR must have the same contents in the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_AIDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0020	MPAMF_AIDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0020	MPAMF_AIDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0020	MPAMF_AIDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0020	MPAMF_AIDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_CCAP_IDR, MPAM Features Cache Capacity Partitioning ID register

The MPAMF_CCAP_IDR characteristics are:

Purpose

Indicates the number of fractional bits in MPAMCFG_CMAX.CMAX.

MPAMF_CCAP_IDR_s indicates the number of fractional bits in the Secure instance of MPAMCFG_CMAX. MPAMF_CCAP_IDR_ns indicates the number of fractional bits in the Non-secure instance of MPAMCFG_CMAX. MPAMF_CCAP_IDR_rt indicates the number of fractional bits in the Root cache capacity control settings register field, MPAMCFG_CMAX.CMAX. MPAMF_CCAP_IDR_rl indicates the number of fractional bits in the Realm cache capacity control settings register field, MPAMCFG_CMAX.CMAX.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selected by MPAMCFG_PART_SEL.RIS. The description of every field that is affected by MPAMCFG_PART_SEL.RIS has information within the field description.

Configuration

The power domain of MPAMF_CCAP_IDR is IMPLEMENTATION DEFINED.

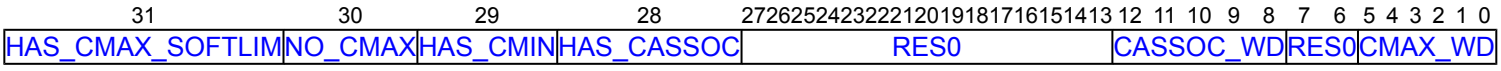
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_CCAP_PART == 1. Otherwise, direct accesses to MPAMF_CCAP_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_CCAP_IDR is a 32-bit register.

Field descriptions



HAS_CMAX_SOFTLIM, bit [31]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Has soft limiting selection field in MPAMCFG_CMAX.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_CMAX_SOFTLIM	Meaning
0b0	If MPAMCFG_CMAX is implemented, it has no SOFTLIM field and the maximum capacity is controlled with a hard limit.
0b1	If MPAMCFG_CMAX is implemented, that register has a SOFTLIMIT field to select between hard or soft limiting to the CMAX parameter.

If RIS is implemented, this field indicates selectable limiting for the cache maximum capacity control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

NO_CMAX, bit [30]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Does not have CMAX partitioning.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NO_CMAX	Meaning
0b0	MPAMCFG_CMAX is implemented.
0b1	MPAMCFG_CMAX is not implemented.

If RIS is implemented, this field indicates the absence of a cache maximum capacity partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_CMIN, bit [29]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Has cache minimum capacity partitioning.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_CMIN	Meaning
0b0	MPAMCFG_CMIN is not implemented.
0b1	MPAMCFG_CMIN is implemented.

If RIS is implemented, this field indicates the presence of a cache minimum capacity partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_CASSOC, bit [28]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Has cache maximum associativity partitioning.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_CASSOC	Meaning
0b0	MPAMCFG_CASSOC is not implemented.
0b1	MPAMCFG_CASSOC is implemented.

If RIS is implemented, this field indicates the presence of a cache maximum associativity partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [27:13]

Reserved, RES0.

CASSOC_WD, bits [12:8]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Number of fractional bits implemented in the cache associativity partitioning control, MPAMCFG_CASSOC.CASSOC, of this MSC. See MPAMCFG_CASSOC.

If RIS is implemented, this field indicates the number of fractional bits in the cache capacity partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [7:6]

Reserved, RES0.

CMAX_WD, bits [5:0]

Number of fractional bits implemented in the cache capacity partitioning control, MPAMCFG_CMAX.CMAX, of this device. See MPAMCFG_CMAX.

This field must contain a value from 1 to 16, inclusive.

If RIS is implemented, this field indicates the number of fractional bits in the cache capacity partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPAMF_CCAP_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_CCAP_IDR is read-only.

MPAMF_CCAP_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_CCAP_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_CCAP_IDR_s is permitted to have either the same or different contents to MPAMF_CCAP_IDR_ns, MPAMF_CCAP_IDR_rt, or MPAMF_CCAP_IDR_rl.
- MPAMF_CCAP_IDR_ns is permitted to have either the same or different contents to MPAMF_CCAP_IDR_rt or MPAMF_CCAP_IDR_rl.
- MPAMF_CCAP_IDR_rt is permitted to have either the same or different contents to MPAMF_CCAP_IDR_rl.

There must be separate registers in the Secure (MPAMF_CCAP_IDR_s), Non-secure (MPAMF_CCAP_IDR_ns), Root (MPAMF_CCAP_IDR_rt), and Realm (MPAMF_CCAP_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_CCAP_IDR shows the configuration of cache capacity partitioning for the cache resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_CCAP_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0038	MPAMF_CCAP_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0038	MPAMF_CCAP_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0038	MPAMF_CCAP_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0038	MPAMF_CCAP_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_CPOR_IDR, MPAM Features Cache Portion Partitioning ID register

The MPAMF_CPOR_IDR characteristics are:

Purpose

Indicates the number of bits in MPAMCFG_CPBM<n>.

MPAMF_CPOR_IDR_s indicates the number of bits in the Secure instance of MPAMCFG_CPBM<n>. MPAMF_CPOR_IDR_ns indicates the number of bits in the Non-secure instance of MPAMCFG_CPBM<n>. MPAMF_CPOR_IDR_rt indicates the number of bits in the Root instance of MPAMCFG_CPBM<n>. MPAMF_CPOR_IDR_rl indicates the number of bits in the Realm instance of MPAMCFG_CPBM<n>.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selector, MPAMCFG_PART_SEL.RIS. The description of every field that is affected by MPAMCFG_PART_SEL.RIS has information within the field description.

Configuration

The power domain of MPAMF_CPOR_IDR is IMPLEMENTATION DEFINED.

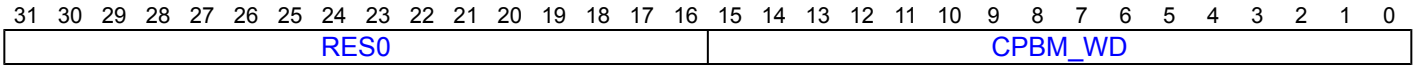
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_CPOR_PART == 1. Otherwise, direct accesses to MPAMF_CPOR_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_CPOR_IDR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

CPBM_WD, bits [15:0]

Number of bits in the cache portion partitioning bit map of this device. See MPAMCFG_CPBM<n>.

This field must contain a value from 1 to 32768, inclusive. Values greater than 32 require a group of 32-bit registers to access the CPBM, up to 1024 if CPBM_WD is the largest value.

If RIS is implemented, this field indicates the number bits in the cache portion bitmap for the resource instance selected by MPAMCFG_PART_SEL.RIS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPAMF_CPOR_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_CPOR_IDR is read-only.

MPAMF_CPOR_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_CPOR_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_CPOR_IDR_s is permitted to have either the same or different contents to MPAMF_CPOR_IDR_ns, MPAMF_CPOR_IDR_rt, or MPAMF_CPOR_IDR_rl.
- MPAMF_CPOR_IDR_ns is permitted to have either the same or different contents to MPAMF_CPOR_IDR_rt or MPAMF_CPOR_IDR_rl.
- MPAMF_CPOR_IDR_rt is permitted to have either the same or different contents to MPAMF_CPOR_IDR_rl.

There must be separate registers in the Secure (MPAMF_CPOR_IDR_s), Non-secure (MPAMF_CPOR_IDR_ns), Root (MPAMF_CPOR_IDR_rt), and Realm (MPAMF_CPOR_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_CPOR_IDR shows the configuration of cache portion partitioning for the cache resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_CPOR_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0030	MPAMF_CPOR_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0030	MPAMF_CPOR_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0030	MPAMF_CPOR_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0030	MPAMF_CPOR_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_CSUMON_IDR, MPAM Features Cache Storage Usage Monitoring ID register

The MPAMF_CSUMON_IDR characteristics are:

Purpose

Indicates the number of cache storage usage monitor instances and other properties of the CSU monitoring.

MPAMF_CSUMON_IDR_s indicates the number and properties of Secure cache storage usage monitoring. MPAMF_CSUMON_IDR_ns indicates the number and properties of Non-secure cache storage usage monitoring. MPAMF_CSUMON_IDR_rt indicates the number and properties of Root cache storage usage monitoring. MPAMF_CSUMON_IDR_rl indicates the number and properties of Realm cache storage usage monitoring.

If MPAMF_IDR.HAS_RIS is 1, fields that mention RIS must reflect the properties of the resource instance currently selected by MPAMCFG_PART_SEL.RIS. Fields that do not mention RIS are constant across all resource instances.

Configuration

The power domain of MPAMF_CSUMON_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MPAMF_CSUMON_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_CSUMON_IDR is a 32-bit register.

Field descriptions

HAS_CAPTURE, bit [31]

The implementation supports copying an MSMON_CSU to the corresponding MSMON_CSU_CAPTURE on a capture event.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_CAPTURE	Meaning
0b0	MSMON_CSU_CAPTURE is not implemented and there is no support for capture events in the CSU monitor.
0b1	The MSMON_CSU_CAPTURE register is implemented and the CSU monitor supports the capture event behavior.

If RIS is implemented, this field indicates that CSU monitor capture is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

CSU_RO, bit [30]

The implementation of MSMON_CSU is read-only.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSU_RO	Meaning
0b0	MSMON_CSU is read/write.
0b1	MSMON_CSU is read-only.

If RIS is implemented, this field indicates that the MSMON_CSU monitor register is read-only for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

HAS_XCL, bit [29]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Has filtering to exclude clean data and implements the MSMON_CFG_CSU_FLT.XCL field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_XCL	Meaning
0b0	MSMON_CFG_CSU_FLT does not implement the XCL field.
0b1	MSMON_CFG_CSU_FLT implements the XCL field to exclude counting data in the clean state in the monitor instance.

If RIS is implemented, this field indicates that the MSMON_CFG_CSU_FLT.XCL field is implemented in the CSU monitor instances for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bit [28]

Reserved, RES0.

HAS_OFLOW_LNKG, bit [27]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Supports MSMON_CFG_CSU_CTL.OFLOW_LNKG field to control how overflow on an instance affects other monitor instances in this MSC.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_OFLOW_LNKG	Meaning
0b0	Does not support CSU overflow linkage.
0b1	Supports CSU overflow linkage and the MSMON_CFG_CSU_CTL.OFLOW_LNKG field.

If RIS is implemented, this field indicates that MSMON_CFG_CSU_CTL.OFLOW_LNKG is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_OFSR, bit [26]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

The CSU monitor overflow status bitmap register, MSMON_CSU_OFSR, is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_OFSR	Meaning
0b0	MSMON_CSU_OFSR register is not implemented.
0b1	MSMON_CSU_OFSR register is implemented.

If RIS is implemented, this field indicates that CSU monitor overflow status bitmap register is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_CEVNT_OFLW, bit [25]
When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Supports MSMON_CFG_CSU_CTL.CEVNT_OFLW field which can enable the CSU monitor instance to perform overflow behaviors on a capture event.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_CEVNT_OFLW	Meaning
0b0	Does not support MSMON_CFG_CSU_CTL.CEVNT_OFLW.
0b1	Supports MSMON_CFG_CSU_CTL.CEVNT_OFLW.

If RIS is implemented, this field indicates that MSMON_CFG_CSU_CTL.CEVNT_OFLW is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_OFLOW_CAPT, bit [24]
When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Supports MSMON_CFG_CSU_CTL.OFLOW_CAPT field which can enable the CSU monitor instance to capture the monitor on an overflow.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_OFLOW_CAPT	Meaning
0b0	Does not support MSMON_CFG_CSU_CTL.OFLOW_CAPT.
0b1	Supports MSMON_CFG_CSU_CTL.OFLOW_CAPT.

If RIS is implemented, this field indicates that MSMON_CFG_CSU_CTL.OFLOW_CAPT is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [23:16]

Reserved, RES0.

NUM_MON, bits [15:0]

The number of cache storage usage monitor instances implemented.

The largest MSMON_CFG_MON_SEL.MON_SEL value is NUM_MON minus 1.

If RIS is implemented, this field indicates the number of CSU monitor instances implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPAMF_CSUMON_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_CSUMON_IDR is read-only.

MPAMF_CSUMON_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_CSUMON_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_CSUMON_IDR_s is permitted to have either the same or different contents to MPAMF_CSUMON_IDR_ns, MPAMF_CSUMON_IDR_rt, or MPAMF_CSUMON_IDR_rl.
- MPAMF_CSUMON_IDR_ns is permitted to have either the same or different contents to MPAMF_CSUMON_IDR_rt or MPAMF_CSUMON_IDR_rl.
- MPAMF_CSUMON_IDR_rt is permitted to have either the same or different contents to MPAMF_CSUMON_IDR_rl.

There must be separate registers in the Secure (MPAMF_CSUMON_IDR_s), Non-secure (MPAMF_CSUMON_IDR_ns), Root (MPAMF_CSUMON_IDR_rt), and Realm (MPAMF_CSUMON_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_CSUMON_IDR shows the configuration of cache storage usage monitoring for the cache resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

Access to MPAMF_CSUMON_IDR is not affected by MSMON_CFG_MON_SEL.RIS.

MPAMF_CSUMON_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0088	MPAMF_CSUMON_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0088	MPAMF_CSUMON_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0088	MPAMF_CSUMON_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0088	MPAMF_CSUMON_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_ECR, MPAM Error Control Register

The MPAMF_ECR characteristics are:

Purpose

MPAMF_ECR is a 32-bit read/write register that controls MPAM error interrupts for this MSC.

MPAMF_ECR_s controls Secure MPAM error handling. MPAMF_ECR_ns controls Non-secure MPAM error handling. MPAMF_ECR_rt controls Root MPAM error handling. MPAMF_ECR_rl controls Realm MPAM error handling.

Configuration

The power domain of MPAMF_ECR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_ECR are RES0.

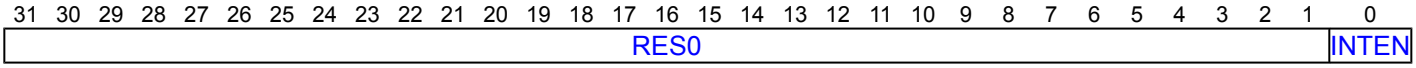
If an MSC cannot encounter any of the error conditions listed in 'Errors in MSCs' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598), both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ECR is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

INTEN, bit [0]

Interrupt Enable.

INTEN	Meaning
0b0	MPAM error interrupts are not signaled.
0b1	MPAM error interrupts are signaled.

Accessing MPAMF_ECR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ECR_s must only be accessible from the Secure MPAM feature page.
- MPAMF_ECR_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMF_ECR_rt must only be accessible from the Root MPAM feature page.
- MPAMF_ECR_rl must only be accessible from the Realm MPAM feature page.

MPAMF_ECR_s, MPAMF_ECR_ns, MPAMF_ECR_rt, and MPAMF_ECR_rl must be separate registers:

- The Secure instance (MPAMF_ECR_s) accesses the error interrupt controls used for Secure PARTIDs.

- The Non-secure instance (MPAMF_ECR_ns) accesses the error interrupt controls used for Non-secure PARTIDs.
- The Root instance (MPAMF_ECR_rt) accesses the error interrupt controls used for Root PARTIDs.
- The Realm instance (MPAMF_ECR_rl) accesses the error interrupt controls used for Realm PARTIDs.

MPAMF_ECR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00F0	MPAMF_ECR_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00F0	MPAMF_ECR_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00F0	MPAMF_ECR_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00F0	MPAMF_ECR_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_ERR_MSI_ADDR_H, MPAM Error MSI High-part Address Register

The MPAMF_ERR_MSI_ADDR_H characteristics are:

Purpose

MPAMF_ERR_MSI_ADDR_H is a 32-bit read/write register for the high part of the MPAM error MSI address.

MPAMF_ERR_MSI_ADDR_H_s is the high part of the MSI write address for error interrupts related to Secure PARTIDs.
MPAMF_ERR_MSI_ADDR_H_ns is the high part of the MSI write address for error interrupts related to Non-secure PARTIDs.
MPAMF_ERR_MSI_ADDR_H_rt is the high part of the MSI write address for error interrupts related to Root PARTIDs.
MPAMF_ERR_MSI_ADDR_H_rl is the high part of the MSI write address for error interrupts related to Realm PARTIDs.

Configuration

The power domain of MPAMF_ERR_MSI_ADDR_H is IMPLEMENTATION DEFINED.

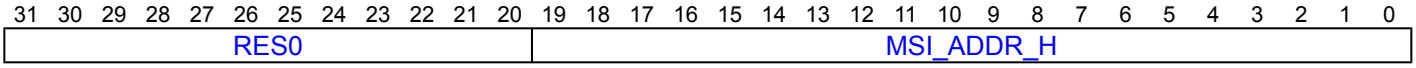
This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_ADDR_H are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_ADDR_H is a 32-bit register.

Field descriptions



Bits [31:20]

Reserved, RES0.

MSI_ADDR_H, bits [19:0]

MSI write address bits[51:32].

Accessing MPAMF_ERR_MSI_ADDR_H

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ERR_MSI_ADDR_H_s must only be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_H_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_H_rt must only be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_ADDR_H_rl must only be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_ADDR_H_s, MPAMF_ERR_MSI_ADDR_H_ns, MPAMF_ERR_MSI_ADDR_H_rt, and MPAMF_ERR_MSI_ADDR_H_rl must be separate registers:

- The Secure instance (MPAMF_ERR_MSI_ADDR_H_s) accesses the high part of the memory address for MSI write to signal an MPAM error used for Secure PARTIDs.

- The Non-secure instance (MPAMF_ERR_MSI_ADDR_H_ns) accesses the high part of the memory address for MSI write to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_ADDR_H_rt) accesses the high part of the memory address for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_ADDR_H_rl) accesses the high part of the memory address for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_ADDR_H can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00E4	MPAMF_ERR_MSI_ADDR_H_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00E4	MPAMF_ERR_MSI_ADDR_H_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00E4	MPAMF_ERR_MSI_ADDR_H_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00E4	MPAMF_ERR_MSI_ADDR_H_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_ERR_MSI_ADDR_L, MPAM Error MSI Low-part Address Register

The MPAMF_ERR_MSI_ADDR_L characteristics are:

Purpose

MPAMF_ERR_MSI_ADDR_L is a 32-bit read/write register for the low part of the MPAM error MSI address.

MPAMF_ERR_MSI_ADDR_L_s is the low part of the MSI write address for error interrupts related to Secure PARTIDs.
MPAMF_ERR_MSI_ADDR_L_ns is the low part of the MSI write address for error interrupts related to Non-secure PARTIDs.
MPAMF_ERR_MSI_ADDR_L_rt is the low part of the MSI write address for error interrupts related to Root PARTIDs.
MPAMF_ERR_MSI_ADDR_L_rl is the low part of the MSI write address for error interrupts related to Realm PARTIDs.

Configuration

The power domain of MPAMF_ERR_MSI_ADDR_L is IMPLEMENTATION DEFINED.

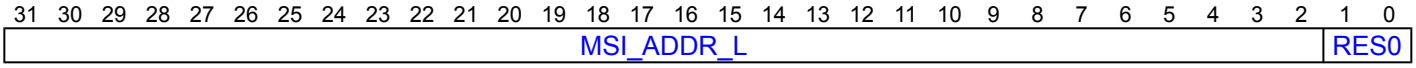
This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_ADDR_L are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_ADDR_L is a 32-bit register.

Field descriptions



MSI_ADDR_L, bits [31:2]

MSI write address bits[31:2].

Bits [1:0]

Reserved, RES0.

Accessing MPAMF_ERR_MSI_ADDR_L

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ERR_MSI_ADDR_L_s must only be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_L_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_L_rt must only be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_ADDR_L_rl must only be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_ADDR_L_s, MPAMF_ERR_MSI_ADDR_L_ns, MPAMF_ERR_MSI_ADDR_L_rt, and MPAMF_ERR_MSI_ADDR_L_rl must be separate registers:

- The Secure instance (MPAMF_ERR_MSI_ADDR_L_s) accesses the low part of the memory address for MSI write to signal an MPAM error used for Secure PARTIDs.

- The Non-secure instance (MPAMF_ERR_MSI_ADDR_L_ns) accesses the low part of the memory address for MSI write to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_ADDR_L_rt) accesses the low part of the memory address for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_ADDR_L_rl) accesses the low part of the memory address for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_ADDR_L can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00E0	MPAMF_ERR_MSI_ADDR_L_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00E0	MPAMF_ERR_MSI_ADDR_L_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00E0	MPAMF_ERR_MSI_ADDR_L_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00E0	MPAMF_ERR_MSI_ADDR_L_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_ERR_MSI_ATTR, MPAM Error MSI Write Attributes Register

The MPAMF_ERR_MSI_ATTR characteristics are:

Purpose

MPAMF_ERR_MSI_ATTR is a 32-bit read/write register that controls MPAM error MSI write attributes for MPAM errors in this MSC.

MPAMF_ERR_MSI_ATTR_s controls the attributes of Secure MPAM error MSI writes. MPAMF_ERR_MSI_ATTR_ns controls the attributes of Non-secure MPAM error MSI writes. MPAMF_ERR_MSI_ATTR_rt controls the attributes of Root MPAM error MSI writes. MPAMF_ERR_MSI_ATTR_rl controls the attributes of Realm MPAM error MSI writes.

Configuration

The power domain of MPAMF_ERR_MSI_ATTR is IMPLEMENTATION DEFINED.

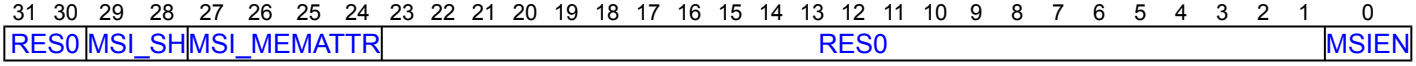
This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_ATTR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_ATTR is a 32-bit register.

Field descriptions



Bits [31:30]

Reserved, RES0.

MSI_SH, bits [29:28]

Sharability attribute of MSI writes.

MSI_SH	Meaning
0b00	Non-shareable.
0b01	Reserved, CONSTRAINED UNPREDICTABLE.
0b10	Outer Shareable.
0b11	Inner Shareable.

When MPAMF_ERR_MSI_ATTR.MSI_MEMATTR specifies a Device memory type, the contents of this field are IGNORED and Shareability is effectively Outer Shareable.

MSI_MEMATTR, bits [27:24]

Memory attributes of MSI writes.

Note

This encoding matches the VMSAv8-64 stage 2 MemAttr[3:0] field as described in the Arm ARM, except that the following encodings are Reserved (not UNPREDICTABLE).

MSI_MEMATTR	Meaning
0b0000	Device-nGnRnE.
0b0001	Device-nGnRE.
0b0010	Device-nGRE.
0b0011	Device-GRE.
0b0100	Reserved. Behave as Device-nGnRnE, 0b0000.
0b0101	Normal Inner Non-cacheable, Outer Non-cacheable.
0b0110	Normal Inner Write-Through Cacheable, Outer Non-cacheable.
0b0111	Normal Inner Write-Back Cacheable, Outer Non-cacheable.
0b1000	Reserved. Behave as Device-nGnRnE, 0b0000.
0b1001	Normal Inner Non-Cachable, Outer Write-Through Cacheable.
0b1010	Normal Inner Write-Through Cacheable, Outer Write-Through Cacheable.
0b1011	Normal Inner Write-Back Cacheable, Outer Write-Through Cacheable.
0b1100	Reserved. Behave as Device-nGnRnE, 0b0000.
0b1101	Normal Inner Non-cacheable, Outer Write-Back Cacheable.
0b1110	Normal Inner Write-Through Cacheable, Outer Write-Back Cacheable.
0b1111	Normal Inner Write-Back Cacheable, Outer Write-Back Cacheable.

When this field specifies a Device memory type, the contents of MPAMF_ERR_MSI_ATTR.MSI_SH are IGNORED and Shareability is effectively Outer Shareable.

Device types may be implemented as any Device type with more than 'n' characters. For example, if this field is set to 0b0010, an implementation may treat the MSI write as the specified type, Device-nGRE, or as Device-nGnRE or as Device-nGnRnE.

Reserved encodings 0b0100, 0b1000, and 0b1100 must be implemented to behave the same as the 0b0000 encoding.

Bits [23:1]

Reserved, RES0.

MSIEN, bit [0]

Error interrupt MSI Enable.

MSIEN	Meaning
0b0	MPAM error MSI writes are not generated to signal enabled MPAM error interrupts. When error MSI writes are disabled, hardwired error interrupts could be generated.
0b1	MPAM error MSI writes are generated to signal enabled MPAM error interrupts. When error MSI writes are enabled, hardwired error interrupts are not generated.

The value of this field affects whether hardwired error interrupts are generated.

The reset behavior of this field is:

- On a MSC reset, this field resets to '0'.

Accessing MPAMF_ERR_MSI_ATTR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ERR_MSI_ATTR_s must only be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_ATTR_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_ATTR_rt must only be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_ATTR_rl must only be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_ATTR_s, MPAMF_ERR_MSI_ATTR_ns, MPAMF_ERR_MSI_ATTR_rt, and MPAMF_ERR_MSI_ATTR_rl must be separate registers:

- The Secure instance (MPAMF_ERR_MSI_ATTR_s) accesses the memory access attributes for MSI write to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_ATTR_ns) accesses the memory access attributes for MSI write to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_ATTR_rt) accesses the memory access attributes for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_ATTR_rl) accesses the memory access attributes for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_ATTR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00EC	MPAMF_ERR_MSI_ATTR_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00EC	MPAMF_ERR_MSI_ATTR_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00EC	MPAMF_ERR_MSI_ATTR_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00EC	MPAMF_ERR_MSI_ATTR_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_ERR_MSI_DATA, MPAM Error MSI Data Register

The MPAMF_ERR_MSI_DATA characteristics are:

Purpose

MPAMF_ERR_MSI_DATA is a 32-bit read/write register for the MPAM error MSI data.

MPAMF_ERR_MSI_DATA_s is the data for the MSI write for error interrupts related to Secure PARTIDs. MPAMF_ERR_MSI_DATA_ns is the data for the MSI write for error interrupts related to Non-secure PARTIDs. MPAMF_ERR_MSI_DATA_rt is the data for the MSI write for error interrupts related to Root PARTIDs. MPAMF_ERR_MSI_DATA_rl is the data for the MSI write for error interrupts related to Realm PARTIDs.

Configuration

The power domain of MPAMF_ERR_MSI_DATA is IMPLEMENTATION DEFINED.

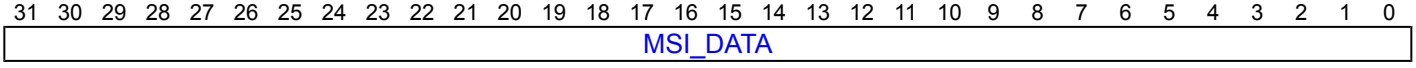
This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_DATA are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_DATA is a 32-bit register.

Field descriptions



MSI_DATA, bits [31:0]

MSI data to be written to ITS to signal an MSI.

Accessing MPAMF_ERR_MSI_DATA

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ERR_MSI_DATA_s must only be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_DATA_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_DATA_rt must only be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_DATA_rl must only be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_DATA_s, MPAMF_ERR_MSI_DATA_ns, MPAMF_ERR_MSI_DATA_rt, and MPAMF_ERR_MSI_DATA_rl must be separate registers:

- The Secure instance (MPAMF_ERR_MSI_DATA_s) accesses the data for MSI write to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_DATA_ns) accesses the data for MSI write to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_DATA_rt) accesses the data for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_DATA_rl) accesses the data for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_DATA can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance

MPAM	MPAMF_BASE_s	0x00E8	MPAMF_ERR_MSI_DATA_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00E8	MPAMF_ERR_MSI_DATA_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00E8	MPAMF_ERR_MSI_DATA_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00E8	MPAMF_ERR_MSI_DATA_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_ERR_MSI_MPAM, MPAM Error MSI Write MPAM Information Register

The MPAMF_ERR_MSI_MPAM characteristics are:

Purpose

MPAMF_ERR_MSI_MPAM is a 32-bit read/write register that sets the MPAM information for error MSI write attributes for MPAM errors in this MSC.

MPAMF_ERR_MSI_MPAM_s controls MPAM information labeling of Secure MPAM error MSI writes. MPAMF_ERR_MSI_MPAM_ns controls MPAM information labeling of Non-secure MPAM error MSI writes. MPAMF_ERR_MSI_MPAM_rt controls MPAM information labeling of Root MPAM error MSI writes. MPAMF_ERR_MSI_MPAM_rl controls MPAM information labeling of Realm MPAM error MSI writes.

Configuration

The power domain of MPAMF_ERR_MSI_MPAM is IMPLEMENTATION DEFINED.

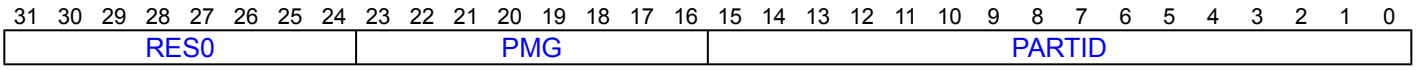
This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_MPAM are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_MPAM is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group property for PARTID MSC error interrupt write.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

PARTID, bits [15:0]

Partition ID for MSC error interrupt write.

The PARTID in this register is in the Secure PARTID space in the MPAMF_ERR_MSI_MPAM_s instance and in the Non-secure PARTID space in the MPAMF_ERR_MSI_MPAM_ns instance of this register.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMF_ERR_MSI_MPAM

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ERR_MSI_MPAM_s must only be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_MPAM_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_MPAM_rt must only be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_MPAM_rl must only be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_MPAM_s, MPAMF_ERR_MSI_MPAM_ns, MPAMF_ERR_MSI_MPAM_rt, and MPAMF_ERR_MSI_MPAM_rl must be separate registers:

- The Secure instance (MPAMF_ERR_MSI_MPAM_s) accesses the MPAM information for MSI write request to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_MPAM_ns) accesses the MPAM information for MSI write request to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_MPAM_rt) accesses the MPAM information for MSI write request to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_MPAM_rl) accesses the MPAM information for MSI write request to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_MPAM can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00DC	MPAMF_ERR_MSI_MPAM_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00DC	MPAMF_ERR_MSI_MPAM_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00DC	MPAMF_ERR_MSI_MPAM_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00DC	MPAMF_ERR_MSI_MPAM_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_ESR, MPAM Error Status Register

The MPAMF_ESR characteristics are:

Purpose

Indicates MPAM error status for this MSC.

MPAMF_ESR_s reports Secure MPAM errors. MPAMF_ESR_ns reports Non-secure MPAM errors. MPAMF_ESR_rt reports Root MPAM errors. MPAMF_ESR_rl reports Realm MPAM errors.

Software should write this register after reading the status of an error to reset ERRCODE to `0x0000` and OVRWR to 0 so that future errors are not reported with OVRWR set.

Configuration

The power domain of MPAMF_ESR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_ESR are RES0.

MPAMF_ESR is 64-bit register when MPAM v0.1 or v1.1 is implemented and MPAMF_IDR.HAS_EXTD_ESR == 1.

Otherwise, MPAMF_ESR is a 32-bit register.

If an MSC cannot encounter any of the error conditions listed in 'Errors in MSCs' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598), both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

The power and reset domain of each MSC component is specific to that component.

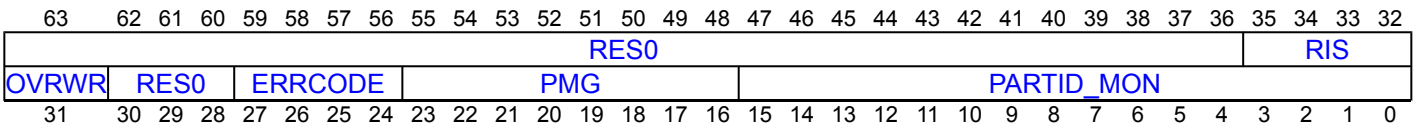
Attributes

MPAMF_ESR is a:

- 64-bit register when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_EXTD_ESR == 1
- 32-bit register otherwise

Field descriptions

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_EXTD_ESR == 1:



Bits [63:36]

Reserved, RES0.

RIS, bits [35:32]

When MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. Where applicable to the ERRCODE, captures the RIS value for the error.

Otherwise:

Reserved, RES0.

OVRWR, bit [31]

Overwritten.

If 0 and ERRCODE == 0b0000, no errors have occurred.

If 0 and ERRCODE is nonzero, a single error has occurred and is recorded in this register.

If 1 and ERRCODE is nonzero, multiple errors have occurred and this register records the most recent error.

The state where this bit is 1 and ERRCODE is zero must not be produced by hardware and is only reached when software writes this combination into this register.

Bits [30:28]

Reserved, RES0.

ERRCODE, bits [27:24]

Error code.

ERRCODE	Meaning
0b0000	No error.
0b0001	PARTID_SEL_Range.
0b0010	Req_PARTID_Range.
0b0011	MSMONCFG_ID_RANGE.
0b0100	Req_PMG_Range.
0b0101	Monitor_Range.
0b0110	intPARTID_Range.
0b0111	Unexpected_INTERNAL.
0b1000	Undefined_RIS_PART_SEL.
0b1001	RIS_No_Control.
0b1010	Undefined_RIS_MON_SEL.
0b1011	RIS_No_Monitor.
0b1100	Reserved.
0b1101	Reserved.
0b1110	Reserved.
0b1111	Reserved.

PMG, bits [23:16]

Program monitoring group.

Set to the PMG on an error that captures PMG. Otherwise, set to 0x00 on an error that does not capture PMG.

PARTID_MON, bits [15:0]

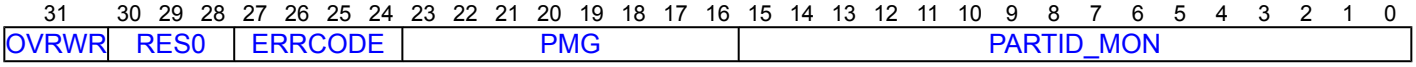
PARTID or monitor.

Set to the PARTID on an error that captures PARTID.

Set to the monitor index on an error that captures MON.

On an error that captures neither PARTID nor MON, this field is set to 0.

Otherwise:



OVRWR, bit [31]

Overwritten.

If 0 and ERRCODE == `0b0000`, no errors have occurred.

If 0 and ERRCODE is nonzero, a single error has occurred and is recorded in this register.

If 1 and ERRCODE is nonzero, multiple errors have occurred and this register records the most recent error.

The state where this bit is 1 and ERRCODE is 0 must not be produced by hardware and is only reached when software writes this combination into this register.

Bits [30:28]

Reserved, RES0.

ERRCODE, bits [27:24]

Error code.

ERRCODE	Meaning
`0b0000`	No error.
`0b0001`	PARTID_SEL_Range.
`0b0010`	Req_PARTID_Range.
`0b0011`	MSMONCFG_ID_RANGE.
`0b0100`	Req_PMG_Range.
`0b0101`	Monitor_Range.
`0b0110`	intPARTID_Range.
`0b0111`	Unexpected_INTERNAL.
`0b1000`	Reserved.
`0b1001`	Reserved.
`0b1010`	Reserved.
`0b1011`	Reserved.
`0b1100`	Reserved.
`0b1101`	Reserved.
`0b1110`	Reserved.
`0b1111`	Reserved.

PMG, bits [23:16]

Program monitoring group.

Set to the PMG on an error that captures PMG. Otherwise, set to `0x00` on an error that does not capture PMG.

PARTID_MON, bits [15:0]

PARTID or monitor.

Set to the PARTID on an error that captures PARTID.

Set to the monitor index on an error that captures MON.

On an error that captures neither PARTID nor MON, this field is set to `0x0000`.

Accessing MPAMF_ESR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ESR_s must only be accessible from the Secure MPAM feature page.
- MPAMF_ESR_ns must only be accessible from the Non-secure MPAM feature page.
- MPAMF_ESR_rt must only be accessible from the Root MPAM feature page.
- MPAMF_ESR_rl must only be accessible from the Realm MPAM feature page.

MPAMF_ESR_s, MPAMF_ESR_ns, MPAMF_ESR_rt, and MPAMF_ESR_rl must be separate registers:

- The Secure instance (MPAMF_ESR_s) accesses the error status used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ESR_ns) accesses the error status used for Non-secure PARTIDs.
- The Root instance (MPAMF_ESR_rt) accesses the error status used for Root PARTIDs.
- The Realm instance (MPAMF_ESR_rl) accesses the error status used for Realm PARTIDs.

MPAMF_ESR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00F8	MPAMF_ESR_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00F8	MPAMF_ESR_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00F8	MPAMF_ESR_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00F8	MPAMF_ESR_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_IDR, MPAM Features Identification Register

The MPAMF_IDR characteristics are:

Purpose

Indicates which memory partitioning and monitoring features are present on this MSC.

MPAMF_IDR_s indicates the MPAM features accessed from the Secure MPAM feature page. MPAMF_IDR_ns indicates the MPAM features accessed from the Non-secure MPAM feature page. MPAMF_IDR_rt indicates the MPAM features accessed from the Root MPAM feature page. MPAMF_IDR_rl indicates the MPAM features accessed from the Realm MPAM feature page.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selected by MPAMCFG_PART_SEL.RIS. The description of every field that is affected by MPAMCFG_PART_SEL.RIS has that information within the field description.

Configuration

The power domain of MPAMF_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_IDR are RES0.

MPAMF_IDR is a 64-bit register when MPAM v0.1 or v1.1 is implemented.

Otherwise, MPAMF_IDR is a 32-bit register.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_IDR is a:

- 64-bit register when FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented
- 32-bit register otherwise

Field descriptions

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

63	62	61	60	59	58	57	56
RES0				RIS_MAX			
HAS_PARTID_NRW	HAS_MSMON	HAS_IMPL_IDR	EXT	HAS_PRI_PART	HAS_MBW_PART	HAS_CPOR_PART	HAS_CCAP_PAR
31	30	29	28	27	26	25	24

Bits [63:60]

Reserved, RES0.

RIS_MAX, bits [59:56]

When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_RIS == 1:

Maximum RIS value supported in MPAMCFG_PART_SEL. Must be `0b0000` if MPAMF_IDR.HAS_RIS == 0.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [55:47]

Reserved, RES0.

HAS_DEFAULT_PARTID, bit [46]

When MPAMF_IDR.EXT == 1:

Indicates support in an MSC for a default resource control configuration.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_DEFAULT_PARTID	Meaning
0b0	The default resource control configuration is supported.
0b1	The default resource control configuration is not supported.

FEAT_MPAM_MSC_DCTRL implements the functionality identified by the value 0b1.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_OUT_TL, bit [45]

When MPAMF_IDR.EXT == 1:

Has egress translation. Indicates that this MSC supports egress MPAM information bundle manipulation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_OUT_TL	Meaning
0b0	MPAM information bundle manipulation for egress translation is not supported.
0b1	MPAM information bundle manipulation for egress translation is supported.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_IN_TL, bit [44]

When MPAMF_IDR.EXT == 1:

Has ingress translation. Indicates that this MSC supports ingress MPAM information bundle manipulation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_IN_TL	Meaning
0b0	MPAM information bundle manipulation for ingress translation is not supported.
0b1	MPAM information bundle manipulation for ingress translation is supported.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_NFU, bit [43]
When MPAMF_IDR.EXT == 1:

Has No Future Use field in MPAMCFG_DIS. Indicates that MPAMCFG_DIS.NFU is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_NFU	Meaning
0b0	MPAMCFG_DIS.NFU is not implemented. A PARTID disabled through access to MPAMCFG_DIS must preserve the control settings of the disabled PARTID.
0b1	Implements MPAMCFG_DIS.NFU. A PARTID disabled with NFU as 1 may have its control settings forgotten.

If MPAMF_IDR.HAS_ENDIS is 0b0, this field must also be 0b0.

This field must be the same in each instance of this register and for any value in MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_ENDIS, bit [42]
When MPAMF_IDR.EXT == 1:

Has PARTID enable and disable. Indicates that this MSC supports PARTID disable and enable via MPAMCFG_DIS, MPAMCFG_EN and MPAMCFG_EN_FLAGS registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_ENDIS	Meaning
0b0	Does not support PARTID enable and disable functionality, and MPAMCFG_EN, MPAMCFG_DIS and MPAMCFG_EN_FLAGS registers are not implemented.
0b1	Supports PARTID enable and disable through the MPAMCFG_EN, MPAMCFG_DIS and MPAMCFG_EN_FLAGS registers.

All three registers must be implemented when this field is 1, MPAMCFG_EN, MPAMCFG_DIS, and MPAMCFG_EN_FLAGS.

This field must be the same in each instance of this register and for any value in MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

SP4, bit [41]
When MPAMF_IDR.EXT == 1 and FEAT_RME is implemented:

Indicates whether this MSC supports 4 PARTID spaces.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SP4	Meaning
0b0	This MSC supports two PARTID spaces.
0b1	This MSC supports four PARTID spaces.

This field must read the same in each instance of this register and for any value in MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_ERR_MSI, bit [40]
When MPAMF_IDR.EXT == 1:

Has support for MSI writes to signal MPAM error interrupts. These registers are implemented: MPAMF_ERR_MSI_ADDR_L, MPAMF_ERR_MSI_ADDR_H, MPAMF_ERR_MSI_ATTR, MPAMF_ERR_MSI_DATA, and MPAMF_ERR_MSI_MPAM.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_ERR_MSI	Meaning
0b0	MPAMF_ERR_MSI_ADDR_L, MPAMF_ERR_MSI_ADDR_H, MPAMF_ERR_MSI_ATTR, MPAMF_ERR_MSI_DATA, and MPAMF_ERR_MSI_MPAM registers are not implemented.
0b1	MPAMF_ERR_MSI_ADDR_L, MPAMF_ERR_MSI_ADDR_H, MPAMF_ERR_MSI_ATTR, MPAMF_ERR_MSI_DATA, and MPAMF_ERR_MSI_MPAM are implemented and can be used to generate writes to signal error interrupts.

If MPAMF_IDR.HAS_ESR is 0, this bit must also be 0.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_ESR, bit [39]
When MPAMF_IDR.EXT == 1:

MPAMF_ESR is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_ESR	Meaning
0b0	MPAMF_ESR, MPAMF_ECR, and MPAM error handling are not implemented.
0b1	MPAMF_ESR, MPAMF_ECR, and MPAM error handling are implemented.

If an MSC cannot encounter any of the error conditions listed in 'Errors in MSCs' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598), both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_EXTD_ESR, bit [38]
When MPAMF_IDR.EXT == 1:

MPAMF_ESR is 64 bits.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_EXTD_ESR	Meaning
0b0	MPAMF_ESR is 32 bits.
0b1	MPAMF_ESR is 64 bits.

When MPAMF_IDR.HAS_RIS and MPAMF_IDR.HAS_ESR, this field must be 1.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

NO_IMPL_MSMON, bit [37]
When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_IMPL_IDR == 1:

MPAMF_IMPL_IDR defines no IMPLEMENTATION DEFINED resource monitors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NO_IMPL_MSMON	Meaning
0b0	MPAMF_IMPL_IDR defines at least one IMPLEMENTATION DEFINED resource monitor.
0b1	MPAMF_IMPL_IDR does not define any IMPLEMENTATION DEFINED resource monitors.

If RIS is implemented, this field indicates the presence of IMPLEMENTATION DEFINED resource monitors described in MPAMF_IMPL_IDR for the selected resource instance.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

NO_IMPL_PART, bit [36]
When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_IMPL_IDR == 1:

MPAMF_IMPL_IDR defines no IMPLEMENTATION DEFINED resource controls.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NO_IMPL_PART	Meaning
0b0	MPAMF_IMPL_IDR defines at least one IMPLEMENTATION DEFINED resource control.
0b1	MPAMF_IMPL_IDR does not define any IMPLEMENTATION DEFINED resource controls.

If RIS is implemented, this field indicates the presence of IMPLEMENTATION DEFINED resource controls described in MPAMF_IMPL_IDR for the selected resource instance.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [35:33]

Reserved, RES0.

HAS_RIS, bit [32]

When MPAMF_IDR.EXT == 1:

Has resource instance selector. Indicates that MPAMCFG_PART_SEL contains the RIS field that selects a resource instance to control.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_RIS	Meaning
0b0	MPAMCFG_PART_SEL does not implement the MPAMCFG_PART_SEL.RIS field or multiple resource instance support.
0b1	MPAMCFG_PART_SEL implements the MPAMCFG_PART_SEL.RIS field and MPAM resource instance numbers up to and including MPAMF_IDR.RIS_MAX.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_PARTID_NRW, bit [31]

Has PARTID narrowing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_PARTID_NRW	Meaning
0b0	Does not have MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID, or intPARTID mapping support.
0b1	Supports the MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID registers.

Access to this field is **RO**.

HAS_MSMON, bit [30]

Has resource Monitors. Indicates whether this MSC has MPAM resource monitors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_MSMON	Meaning
0b0	Does not support MPAM resource monitoring by groups or MPAMF_MSMON_IDR.
0b1	Supports resource monitoring by matching a combination of PARTID and PMG. See MPAMF_MSMON_IDR.

Access to this field is **RO**.

HAS_IMPL_IDR, bit [29]

Has MPAMF_IMPL_IDR. Indicates whether this MSC has the IMPLEMENTATION SPECIFIC MPAM features register, MPAMF_IMPL_IDR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_IMPL_IDR	Meaning
0b0	Does not have MPAMF_IMPL_IDR.
0b1	Has MPAMF_IMPL_IDR.

Access to this field is **RO**.

EXT, bit [28]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Extended MPAMF_IDR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXT	Meaning
0b0	MPAMF_IDR has no defined bits in [63:32]. The register is effectively 32 bits.
0b1	MPAMF_IDR has bits defined in [63:32]. The register is 64-bits.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_PRI_PART, bit [27]

Has Priority Partitioning. Indicates that MPAM priority partitioning is implemented and MPAMF_PRI_IDR exists.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_PRI_PART	Meaning
0b0	Does not support priority partitioning or have MPAMF_PRI_IDR.
0b1	Has priority partitioning and MPAMF_PRI_IDR.

If RIS is implemented, this field indicates the presence of priority partitioning resource controls as described in MPAMF_PRI_IDR for the selected resource instance.

Access to this field is **RO**.

HAS_MBW_PART, bit [26]

Has Memory Bandwidth Partitioning. Indicates whether this MSC implements MPAM memory bandwidth partitioning and MPAMF_MBW_IDR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_MBW_PART	Meaning
0b0	Does not support memory bandwidth partitioning or have MPAMF_MBW_IDR register.
0b1	Has MPAMF_MBW_IDR register.

If RIS is implemented, this field indicates the presence of memory bandwidth partitioning resource controls as described in MPAMF_MBW_IDR for the selected resource instance.

Access to this field is **RO**.

HAS_CPOR_PART, bit [25]

Has Cache Portion Partitioning. Indicates whether this MSC implements MPAM cache portion partitioning and MPAMF_CPOR_IDR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_CPOR_PART	Meaning
0b0	Does not support cache portion partitioning or have MPAMF_CPOR_IDR or MPAMCFG_CPBM<n> registers.
0b1	Has MPAMF_CPOR_IDR and MPAMCFG_CPBM<n> registers.

If RIS is implemented, this field indicates the presence of cache portion partitioning resource controls as described in MPAMF_CPOR_IDR for the selected resource instance.

Access to this field is **RO**.

HAS_CCAP_PART, bit [24]

Has Cache Capacity Partitioning. Indicates whether this MSC implements MPAM cache capacity partitioning and the MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_CCAP_PART	Meaning
0b0	Does not support cache capacity partitioning or have MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.
0b1	Has MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

If RIS is implemented, this field indicates the presence of cache capacity partitioning resource controls as described in MPAMF_CPOR_IDR for the selected resource instance.

Access to this field is **RO**.

PMG_MAX, bits [23:16]

Maximum supported value of PMG.

The value of this field is permitted to vary between the instances of MPAMF_IDR, each reporting the maximum supported PMG value in the PARTID space associated with that instance.

In MPAMF_IDR_s, this field is permitted to report the maximum PMG value for the Non-secure PARTID space or for the Secure PARTID space. The maximum PMG value for the Secure PARTID space can be read from MPAMF_SIDR.PMG_MAX.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PARTID_MAX, bits [15:0]

Maximum supported value of PARTID.

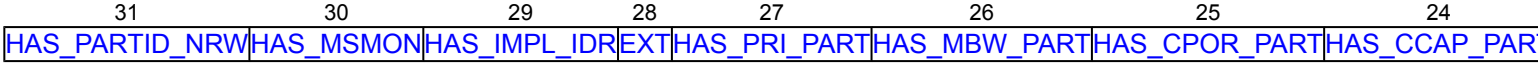
The value of this field is permitted to vary between the instances of MPAMF_IDR, each reporting the maximum supported PARTID value in the PARTID space associated with that instance.

In MPAMF_IDR_s, this field is permitted to report the maximum PARTID value for the Non-secure PARTID space or for the Secure PARTID space. The maximum PARTID value for the Secure PARTID space can be read from MPAMF_SIDR.PARTID_MAX.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:



HAS_PARTID_NRW, bit [31]

Has PARTID Narrowing.

HAS_PARTID_NRW	Meaning
0b0	Does not have MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID, or intPARTID mapping support.
0b1	Supports the MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID registers.

HAS_MSMON, bit [30]

Has resource Monitors. Indicates whether this MSC has MPAM resource monitors.

HAS_MSMON	Meaning
0b0	Does not support MPAM resource monitoring by groups or MPAMF_MSMON_IDR.
0b1	Supports resource monitoring by matching a combination of PARTID and PMG. See MPAMF_MSMON_IDR.

HAS_IMPL_IDR, bit [29]

Has MPAMF_IMPL_IDR. Indicates whether this MSC has the IMPLEMENTATION SPECIFIC MPAM features register, MPAMF_IMPL_IDR.

HAS_IMPL_IDR	Meaning
0b0	Does not have MPAMF_IMPL_IDR.
0b1	Has MPAMF_IMPL_IDR.

EXT, bit [28]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Extended MPAMF_IDR.

EXT	Meaning
0b0	MPAMF_IDR has no defined bits in [63:32]. The register is effectively 32 bits.
0b1	MPAMF_IDR has bits defined in [63:32]. The register is 64-bits.

Otherwise:

Reserved, RES0.

HAS_PRI_PART, bit [27]

Has Priority Partitioning. Indicates whether this MSC implements MPAM priority partitioning and MPAMF_PRI_IDR.

HAS_PRI_PART	Meaning
0b0	Does not support priority partitioning or have MPAMF_PRI_IDR.
0b1	Has MPAMF_PRI_IDR.

HAS_MBW_PART, bit [26]

Has Memory Bandwidth Partitioning. Indicates whether this MSC implements MPAM memory bandwidth partitioning and MPAMF_MBW_IDR.

HAS_MBW_PART	Meaning
0b0	Does not support memory bandwidth partitioning or have MPAMF_MBW_IDR register.
0b1	Has MPAMF_MBW_IDR register.

HAS_CPOR_PART, bit [25]

Has Cache Portion Partitioning. Indicates whether this MSC implements MPAM cache portion partitioning and MPAMF_CPOR_IDR.

HAS_CPOR_PART	Meaning
0b0	Does not support cache portion partitioning or have MPAMF_CPOR_IDR or MPAMCFG_CPBM<n> registers.
0b1	Has MPAMF_CPOR_IDR and MPAMCFG_CPBM<n> registers.

HAS_CCAP_PART, bit [24]

Has Cache Capacity Partitioning. Indicates whether this MSC implements MPAM cache capacity partitioning and the MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

HAS_CCAP_PART	Meaning
0b0	Does not support cache capacity partitioning or have MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.
0b1	Has MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

PMG_MAX, bits [23:16]

Maximum supported value of PMG.

The value of this field is permitted to vary between the instances of MPAMF_IDR, each reporting the maximum supported PMG value in the PARTID space associated with that instance.

In MPAMF_IDR_s this field is permitted to report the maximum PMG value for the Non-secure PARTID space or for the Secure PARTID space. The maximum PMG value for the Secure PARTID space can be read from MPAMF_SIDR.PMG_MAX.

PARTID_MAX, bits [15:0]

Maximum supported value of PARTID.

The value of this field is permitted to vary between the instances of MPAMF_IDR, each reporting the maximum supported PARTID value in the PARTID space associated with that instance.

In MPAMF_IDR_s this field is permitted to report the maximum PARTID value for the Non-secure PARTID space or for the Secure PARTID space. The maximum PARTID value for the Secure PARTID space can be read from MPAMF_SIDR.PARTID_MAX.

Accessing MPAMF_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_IDR is read-only.

MPAMF_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_IDR_s is permitted to have either the same or different contents to MPAMF_IDR_ns, MPAMF_IDR_rt, or MPAMF_IDR_rl.
- MPAMF_IDR_ns is permitted to have either the same or different contents to MPAMF_IDR_rt or MPAMF_IDR_rl.
- MPAMF_IDR_rt is permitted to have either the same or different contents to MPAMF_IDR_rl.

There must be separate registers in the Secure (MPAMF_IDR_s), Non-secure (MPAMF_IDR_ns), Root (MPAMF_IDR_rt), and Realm (MPAMF_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_IDR shows the configuration of MSC MPAM for the resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0000	MPAMF_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0000	MPAMF_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0000	MPAMF_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0000	MPAMF_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_IIDR, MPAM Implementation Identification Register

The MPAMF_IIDR characteristics are:

Purpose

Uniquely identifies the MSC implementation by the combination of implementer, product ID, variant, and revision.

Configuration

The power domain of MPAMF_IIDR is IMPLEMENTATION DEFINED.

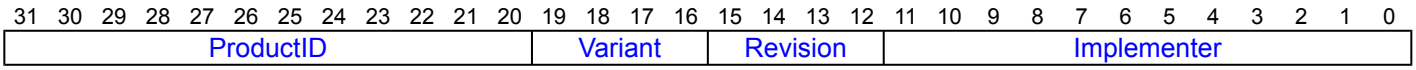
This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_IIDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_IIDR is a 32-bit register.

Field descriptions



ProductID, bits [31:20]

The MSC implementer as identified in the MPAMF_IIDR.Implementer field must assure each product has a unique ProductID from any other with the same Implementer value.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [19:16]

This field distinguishes product variants or major revisions of the product.

Note

Implementations of ProductID with differing software interfaces are expected to have different values in the MPAMF_IIDR.Variant field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

This field distinguishes minor revisions of the product.

Note

This field is intended to differentiate product revisions that are minor changes and are largely software compatible with previous revisions.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the MPAM MSC.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for MPAMF_IIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPAMF_IIDR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_IIDR is read-only.

MPAMF_IIDR must be readable from the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_IIDR must have the same contents in the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_IIDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0018`	MPAMF_IIDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	`0x0018`	MPAMF_IIDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0018`	MPAMF_IIDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0018`	MPAMF_IIDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_IMPL_IDR, MPAM Implementation-Specific Partitioning Feature Identification Register

The MPAMF_IMPL_IDR characteristics are:

Purpose

Indicates the implementation-defined partitioning and monitoring features and parameters of the MSC.

MPAMF_IMPL_IDR_s indicates IMPLEMENTATION DEFINED partitioning and monitoring features accessed from the Secure MPAM feature page. MPAMF_IMPL_IDR_ns indicates those accessed from the Non-secure MPAM feature page. MPAMF_IMPL_IDR_rt indicates IMPLEMENTATION DEFINED partitioning and monitoring features accessed from the Root MPAM feature page. MPAMF_IMPL_IDR_rl indicates those accessed from the Realm MPAM feature page.

If MPAMF_IDR.HAS_RIS is 1, this register gives the implementation-specific features and parameters of the resource instance selected by MPAMCFG_PART_SEL.RIS for any features that are specific to the resource.

Configuration

The power domain of MPAMF_IMPL_IDR is IMPLEMENTATION DEFINED.

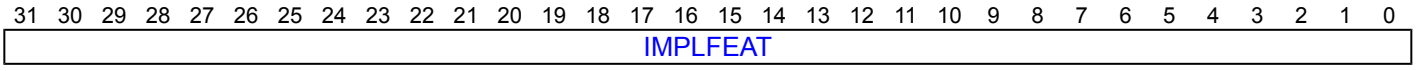
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_IMPL_IDR == 1. Otherwise, direct accesses to MPAMF_IMPL_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_IMPL_IDR is a 32-bit register.

Field descriptions



IMPLFEAT, bits [31:0]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

IMPLFEAT, bits [31:0] of bits [31:0]

All 32 bits of this register are available to be used as the implementer sees fit to indicate the presence of IMPLEMENTATION DEFINED MPAM features in this MSC and to give additional implementation-specific read-only information about the parameters of implementation-specific MPAM features to software.

If RIS is implemented, this register indicates the implementation-specific features and parameters of the resource instance selected by MPAMCFG_PART_SEL.RIS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

IMPLEMENTATION DEFINED, bits [31:0] of bits [31:0]

All 32 bits of this register are available to be used as the implementer sees fit to indicate the presence of IMPLEMENTATION DEFINED MPAM features in this MSC and to give additional implementation-specific read-only information about the parameters of implementation-specific MPAM features to software.

Accessing MPAMF_IMPL_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_IMPL_IDR is read-only.

MPAMF_IMPL_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_IMPL_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_IMPL_IDR_s is permitted to have either the same or different contents to MPAMF_IMPL_IDR_ns, MPAMF_IMPL_IDR_rt, or MPAMF_IMPL_IDR_rl.
- MPAMF_IMPL_IDR_ns is permitted to have either the same or different contents to MPAMF_IMPL_IDR_rt or MPAMF_IMPL_IDR_rl.
- MPAMF_IMPL_IDR_rt is permitted to have either the same or different contents to MPAMF_IMPL_IDR_rl.

There must be separate registers in the Secure (MPAMF_IMPL_IDR_s), Non-secure (MPAMF_IMPL_IDR_ns), Root (MPAMF_IMPL_IDR_rt), and Realm (MPAMF_IMPL_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_IMPL_IDR shows the configuration of implementation-specific features for the resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_IMPL_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0028	MPAMF_IMPL_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0028	MPAMF_IMPL_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0028	MPAMF_IMPL_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0028	MPAMF_IMPL_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_IN_TL_IDR, MPAM Ingress PARTID Translation ID Register

The MPAMF_IN_TL_IDR characteristics are:

Purpose

Indicates the ingress PARTID translation capabilities of the MSC.

Configuration

The power domain of MPAMF_IN_TL_IDR is IMPLEMENTATION DEFINED.

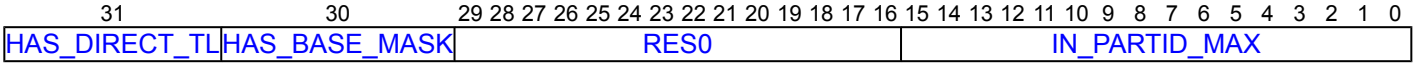
This register is present only when FEAT_MPAM_MSC_DOMAINS is implemented and MPAMF_IDR.HAS_IN_TL == 1. Otherwise, direct accesses to MPAMF_IN_TL_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_IN_TL_IDR is a 32-bit register.

Field descriptions



HAS_DIRECT_TL, bit [31]

Indicates support for direct ingress translation of PARTIDs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_DIRECT_TL	Meaning
0b0	Direct ingress translation of PARTIDs is not supported.
0b1	Direct ingress translation of PARTIDs is supported for those PARTIDs with an explicitly set translation configuration.

Access to this field is **RO**.

HAS_BASE_MASK, bit [30]

Indicates support for computed ingress translation of PARTIDs using a configurable mask and base.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_BASE_MASK	Meaning
0b0	Computed ingress translation of PARTIDs using a configurable mask and base is not supported.
0b1	Computed ingress translation of PARTIDs using a configurable mask and base is supported for those PARTIDs without an explicitly set translation configuration.

Access to this field is **RO**.

Bits [29:16]

Reserved, RES0.

IN_PARTID_MAX, bits [15:0]

When MPAMF_IN_TL_IDR.HAS_DIRECT_TL == 1:

Maximum value for PARTIDs to be used as direct ingress translations, as configured in MPAMCFG_IN_TL.PARTID_TL.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing MPAMF_IN_TL_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_IN_TL_IDR is read-only.

MPAMF_IN_TL_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_IN_TL_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_IN_TL_IDR_s is permitted to have either the same or different contents to MPAMF_IN_TL_IDR_ns, MPAMF_IN_TL_IDR_rt, or MPAMF_IN_TL_IDR_rl.
- MPAMF_IN_TL_IDR_ns is permitted to have either the same or different contents to MPAMF_IN_TL_IDR_rt or MPAMF_IN_TL_IDR_rl.
- MPAMF_IN_TL_IDR_rt is permitted to have either the same or different contents to MPAMF_IN_TL_IDR_rl.

There must be separate registers in the Secure (MPAMF_IN_TL_IDR_s), Non-secure (MPAMF_IN_TL_IDR_ns), Root (MPAMF_IN_TL_IDR_rt), and Realm (MPAMF_IN_TL_IDR_rl) MPAM feature pages.

MPAMF_IN_TL_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x3000	MPAMF_IN_TL_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x3000	MPAMF_IN_TL_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x3000	MPAMF_IN_TL_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x3000	MPAMF_IN_TL_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_MBW_IDR, MPAM Memory Bandwidth Partitioning Identification Register

The MPAMF_MBW_IDR characteristics are:

Purpose

Indicates which MPAM bandwidth partitioning features are present on this MSC.

MPAMF_MBW_IDR_s indicates bandwidth partitioning features accessed from the Secure MPAM feature page. MPAMF_MBW_IDR_ns indicates bandwidth partitioning features accessed from the Non-secure MPAM feature page. MPAMF_MBW_IDR_rt indicates bandwidth partitioning features accessed from the Root MPAM feature page. MPAMF_MBW_IDR_rl indicates bandwidth partitioning features accessed from the Realm MPAM feature page.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selected by MPAMCFG_PART_SEL.RIS. The description of every field that is affected by MPAMCFG_PART_SEL.RIS has that information within the field description.

Configuration

The power domain of MPAMF_MBW_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_MBW_PART == 1. Otherwise, direct accesses to MPAMF_MBW_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_MBW_IDR is a 32-bit register.

Field descriptions

Bits [31:29]

Reserved, RES0.

BWPBM_WD, bits [28:16]

Bandwidth portion bitmap width.

The number of bandwidth portion bits in the MPAMCFG_MBW_PBM<n> register array.

If MPAMF_MBW_IDR.HAS_PBM is 1, this field must contain a value from 1 to 4096, inclusive. Values greater than 32 require a group of 32-bit registers to access the BWPBM, up to 128 if BWPBM_WD is the largest value.

If MPAMF_MBW_IDR.HAS_PBM is 0, this field must be ignored by software.

If RIS is implemented, this field indicates the width of the memory bandwidth portion bitmap partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [15]

Reserved, RES0.

WINDWR, bit [14]

Indicates the bandwidth accounting period register is writable.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WINDWR	Meaning
0b0	The bandwidth accounting period is readable from MPAMCFG_MBW_WINWD which might be fixed or vary due to clock rate reconfiguration of the memory channel or memory controller.
0b1	The bandwidth accounting width is readable and writable per partition in MPAMCFG_MBW_WINWD.

Access to this field is **RO**.

HAS_PROP, bit [13]

Indicates that this MSC implements proportional stride bandwidth partitioning and the MPAMCFG_MBW_PROP register can be accessed.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_PROP	Meaning
0b0	There is no memory bandwidth proportional stride control and the MPAMCFG_MBW_PROP register is RES0.
0b1	The proportional stride memory bandwidth partitioning scheme is supported and the MPAMCFG_MBW_PROP register can be accessed.

If RIS is implemented, this field indicates the presence of the memory bandwidth proportional stride partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

HAS_PBM, bit [12]

Indicates that bandwidth portion partitioning is implemented and the MPAMCFG_MBW_PBM<n> register array can be accessed.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_PBM	Meaning
0b0	There is no memory bandwidth portion control and the MPAMCFG_MBW_PBM<n> is RES0.
0b1	The memory bandwidth portion allocation scheme exists and the MPAMCFG_MBW_PBM<n> register can be accessed.

If RIS is implemented, this field indicates the presence of the memory bandwidth portion partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

HAS_MAX, bit [11]

Indicates that this MSC implements maximum bandwidth partitioning and the MPAMCFG_MBW_MAX register can be accessed.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_MAX	Meaning
0b0	There is no maximum memory bandwidth control and the MPAMCFG_MBW_MAX register is RES0.
0b1	The maximum memory bandwidth allocation scheme is supported and the MPAMCFG_MBW_MAX register can be accessed. The MPAMF_MBW_IDR.MAX_LIM and MPAMCFG_MBW_MAX.HARDLIM fields indicate which limit behaviors are implemented and enabled.

If RIS is implemented, this field indicates the presence of the maximum bandwidth partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

HAS_MIN, bit [10]

Indicates that this MSC implements minimum bandwidth partitioning and the MPAMCFG_MBW_MIN register can be accessed.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_MIN	Meaning
0b0	There is no minimum memory bandwidth control and the MPAMCFG_MBW_MIN register is RES0.
0b1	The minimum memory bandwidth allocation scheme is supported and the MPAMCFG_MBW_MIN register can be accessed.

If RIS is implemented, this field indicates the presence of the minimum bandwidth partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

MAX_LIM, bits [9:8]

Implemented maximum bandwidth partitioning behaviors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MAX_LIM	Meaning
0b00	Both soft limit and hard limit behaviors are implemented.
0b01	Soft limit behavior is implemented, hard limit behavior is not implemented.
0b10	Hard limit behavior is implemented, soft limit behavior is not implemented.

If both soft limit and hard limit behaviors are implemented, MPAMCFG_MBW_MAX.HARDLIM selects which behavior is used.

If RIS is implemented, this field indicates the implemented maximum bandwidth limit partitioning behaviors for the resource instance selected by MPAMCFG_PART_SEL.RIS.

When MPAMF_MBW_IDR.HAS_MAX != 1, access to this field is **RES0**.

Bits [7:6]

Reserved, RES0.

BWA_WD, bits [5:0]

Number of implemented bits in the bandwidth allocation fields: MIN, MAX, and STRIDE. See MPAMCFG_MBW_MIN, MPAMCFG_MBW_MAX, and MPAMCFG_MBW_PROP.

In any of these bandwidth allocation fields exist, this field must have a value from 1 to 16, inclusive. Otherwise, it is permitted to be 0.

If RIS is implemented, this field indicates the number of implemented bits in the bandwidth allocation control fields for the resource instance selected by MPAMCFG_PART_SEL.RIS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPAMF_MBW_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_MBW_IDR is read-only.

MPAMF_MBW_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_MBW_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_MBW_IDR_s is permitted to have either the same or different contents to MPAMF_MBW_IDR_ns, MPAMF_MBW_IDR_rt, or MPAMF_MBW_IDR_rl.
- MPAMF_MBW_IDR_ns is permitted to have either the same or different contents to MPAMF_MBW_IDR_rt or MPAMF_MBW_IDR_rl.
- MPAMF_MBW_IDR_rt is permitted to have either the same or different contents to MPAMF_MBW_IDR_rl.

There must be separate registers in the Secure (MPAMF_MBW_IDR_s), Non-secure (MPAMF_MBW_IDR_ns), Root (MPAMF_MBW_IDR_rt), and Realm (MPAMF_MBW_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_MBW_IDR shows the configuration of memory bandwidth partitioning for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_MBW_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0040	MPAMF_MBW_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0040	MPAMF_MBW_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0040	MPAMF_MBW_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0040	MPAMF_MBW_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_MBWUMON_IDR, MPAM Features Memory Bandwidth Usage Monitoring ID register

The MPAMF_MBWUMON_IDR characteristics are:

Purpose

Indicates the number of memory bandwidth usage monitor instances implemented. This register also indicates several properties of MBWU monitoring, including whether the implementation supports capture, scaling, or long counters.

MPAMF_MBWUMON_IDR_s indicates the number of Secure memory bandwidth usage monitor instances. MPAMF_MBWUMON_IDR_ns indicates the number of Non-secure memory bandwidth usage monitor instances. MPAMF_MBWUMON_IDR_rt indicates the number of Root memory bandwidth usage monitor instances. MPAMF_MBWUMON_IDR_rl indicates the number of Realm memory bandwidth usage monitor instances.

If MPAMF_IDR.HAS_RIS is 1, fields that mention RIS must reflect the properties of the resource instance currently selected by MPAMCFG_PART_SEL.RIS. Fields that do not mention RIS are constant across all resource instances.

Configuration

The power domain of MPAMF_MBWUMON_IDR is IMPLEMENTATION DEFINED.

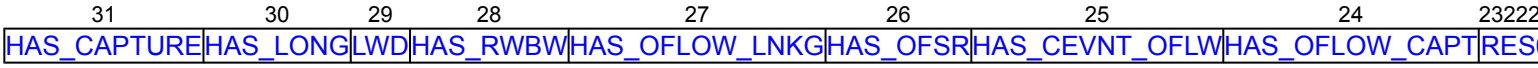
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MPAMF_MBWUMON_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_MBWUMON_IDR is a 32-bit register.

Field descriptions



HAS_CAPTURE, bit [31]

The implementation supports copying an MSMON_MBWU to the corresponding MSMON_MBWU_CAPTURE on a capture event.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_CAPTURE	Meaning
0b0	MSMON_MBWU_CAPTURE is not implemented and there is no support for capture events in the MBWU monitor.
0b1	The MSMON_MBWU_CAPTURE register is implemented and the MBWU monitor supports the capture event behavior.

If RIS is implemented, this field indicates that MBWU monitor capture is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

If MPAMF_MBWUMON_IDR.HAS_LONG is 1, this also indicates that MSMON_MBWU_L_CAPTURE is implemented.

Access to this field is **RO**.

HAS_LONG, bit [30]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Indicates whether MSMON_MBWU_L is implemented.

If HAS_CAPTURE is 1, indicates whether MSMON_MBWU_L_CAPTURE is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_LONG	Meaning
0b0	Does not implement MSMON_MBWU_L or MSMON_MBWU_L_CAPTURE.
0b1	Implements MSMON_MBWU_L. If HAS_CAPTURE == 1, MSMON_MBWU_L_CAPTURE is also implemented.

If RIS is implemented, this field indicates that the long MBWU monitor is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

If MPAMF_MBWUMON_IDR.HAS_CAPTURE is 1, this also indicates that MSMON_MBWU_L_CAPTURE is implemented.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

LWD, bit [29]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Long register VALUE width.

If MPAMF_MBWUMON_IDR.HAS_LONG is 0, MPAMF_MBWUMON_IDR.LWD must also be 0.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LWD	Meaning
0b0	If MPAMF_MBWUMON_IDR.HAS_LONG is 1, MSMON_MBWU_L has 44-bit VALUE field in bits [43:0]. Bits [62:44] are RES0. If HAS_LONG is 1 and MPAMF_MBWUMON_IDR.HAS_CAPTURE is 1, MSMON_MBWU_L_CAPTURE also has 44-bit VALUE field in bits [43:0].
0b1	MSMON_MBWU_L has 63-bit VALUE field in bits [62:0]. If MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1, MSMON_MBWU_L_CAPTURE also has 63-bit VALUE field in bits [62:0].

If RIS is implemented, this field indicates the length of the MSMON_MBWU_L.VALUE field implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_RWBW, bit [28]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Read/write bandwidth selection is implemented in MSMON_CFG_MBWU_FLT.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_RWBW	Meaning
0b0	Read/write bandwidth selection is not implemented.
0b1	Read/write bandwidth selection is implemented.

If RIS is implemented, this field indicates whether read/write bandwidth collection selection is available in MSMON_CFG_MBWU_FLT for resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_OFLOW_LNKG, bit [27]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Supports MSMON_CFG_MBWU_CTL.OFLOW_LNKG field to control how overflow on an instance affects other monitor instances in this MSC.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_OFLOW_LNKG	Meaning
0b0	Does not support MBWU overflow linkage.
0b1	Supports MBWU overflow linkage and the MSMON_CFG_MBWU_CTL.OFLOW_LNKG field.

If RIS is implemented, this field indicates that MSMON_CFG_MBWU_CTL.OFLOW_LNKG is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_OFSR, bit [26]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

The MBWU monitor overflow status bitmap register, MSMON_MBWU_OFSR, is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_OFSR	Meaning
0b0	MSMON_MBWU_OFSR register is not implemented.
0b1	MSMON_MBWU_OFSR register is implemented.

If RIS is implemented, this field indicates that MBWU monitor overflow status bitmap register is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_CEVNT_OFLW, bit [25]

Supports MSMON_CFG_MBWU_CTL.CEVNT_OFLW field which can enable the MBWU monitor instance to perform overflow behaviors on a capture event.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_CEVNT_OFLW	Meaning
0b0	Does not support MSMON_CFG_MBWU_CTL.CEVNT_OFLW.
0b1	Supports MSMON_CFG_MBWU_CTL.CEVNT_OFLW.

If RIS is implemented, this field indicates that MSMON_CFG_MBWU_CTL.CEVNT_OFLW is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

HAS_OFLOW_CAPT, bit [24]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Supports MSMON_CFG_MBWU_CTL.OFLOW_CAPT field which can enable the MBWU monitor instance to capture the monitor on an overflow.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_OFLOW_CAPT	Meaning
0b0	Does not support MSMON_CFG_MBWU_CTL.OFLOW_CAPT.
0b1	Supports MSMON_CFG_MBWU_CTL.OFLOW_CAPT.

If RIS is implemented, this field indicates that MSMON_CFG_MBWU_CTL.OFLOW_CAPT is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [23:21]

Reserved, RES0.

SCALE, bits [20:16]

Scaling of MSMON_MBWU.VALUE in bits. If scaling is enabled by MSMON_CFG_MBWU_CTL.SCLEN, the byte count in the VALUE field has been shifted by SCALE bits to the right.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SCALE	Meaning
0b00000	Scaling is not implemented.
0bxxxxx	Other values are right shift count when scaling is enabled.

If RIS is implemented, this field indicates the scale value for MSMON_MBWU.VALUE field for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

NUM_MON, bits [15:0]

The number of memory bandwidth usage monitor instances implemented. The largest monitor instance selector, MSMON_CFG_MON_SEL.MON_SEL, is NUM_MON minus 1.

If RIS is implemented, this field indicates the number of MBWU monitor instances for MSMON_MBWU.VALUE field for the resource instance selected by MPAMCFG_PART_SEL.RIS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPAMF_MBWUMON_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_MBWUMON_IDR is read-only.

MPAMF_MBWUMON_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_MBWUMON_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_MBWUMON_IDR_s is permitted to have either the same or different contents to MPAMF_MBWUMON_IDR_ns, MPAMF_MBWUMON_IDR_rt, or MPAMF_MBWUMON_IDR_rl.
- MPAMF_MBWUMON_IDR_ns is permitted to have either the same or different contents to MPAMF_MBWUMON_IDR_rt or MPAMF_MBWUMON_IDR_rl.
- MPAMF_MBWUMON_IDR_rt is permitted to have either the same or different contents to MPAMF_MBWUMON_IDR_rl.

There must be separate registers in the Secure (MPAMF_MBWUMON_IDR_s), Non-secure (MPAMF_MBWUMON_IDR_ns), Root (MPAMF_MBWUMON_IDR_rt), and Realm (MPAMF_MBWUMON_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_MBWUMON_IDR shows the configuration of memory bandwidth monitoring for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

Access to MPAMF_MBWUMON_IDR is not affected by MSMON_CFG_MON_SEL.RIS.

MPAMF_MBWUMON_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0090	MPAMF_MBWUMON_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0090	MPAMF_MBWUMON_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0090	MPAMF_MBWUMON_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0090	MPAMF_MBWUMON_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_MSMON_IDR, MPAM Resource Monitoring Identification Register

The MPAMF_MSMON_IDR characteristics are:

Purpose

Indicates which MPAM monitoring features are present on this MSC.

MPAMF_MSMON_IDR_s indicates Secure monitoring features. MPAMF_MSMON_IDR_ns indicates Non-secure monitoring features. MPAMF_MSMON_IDR_rt indicates Root monitoring features. MPAMF_MSMON_IDR_rl indicates Realm monitoring features.

If MPAMF_IDR.HAS_RIS is 1, fields that mention RIS must reflect the properties of the resource instance currently selected by MPAMCFG_PART_SEL.RIS. Fields that do not mention RIS are constant across all resource instances.

Configuration

The power domain of MPAMF_MSMON_IDR is IMPLEMENTATION DEFINED.

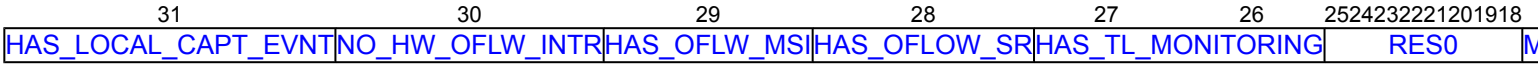
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_MSMON == 1. Otherwise, direct accesses to MPAMF_MSMON_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_MSMON_IDR is a 32-bit register.

Field descriptions



HAS_LOCAL_CAPT_EVNT, bit [31]

Has local capture event generator. Indicates whether this MSC has the MPAM local capture event generator and the MSMON_CAPT_EVNT register.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_LOCAL_CAPT_EVNT	Meaning
0b0	Does not support MPAM local capture event generator or MSMON_CAPT_EVNT.
0b1	Supports the MPAM local capture event generator and the MSMON_CAPT_EVNT register.

Access to this field is **RO**.

NO_HW_OFLW_INTR, bit [30]

When FEAT_MPAMv1p1 is implemented:

Does not have hardwired MPAM monitor overflow interrupt.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NO_HW_OFLW_INTR	Meaning
0b0	Supports generating a hardwired interrupt to signal MPAM monitor overflow.
0b1	No support for a hardwired interrupt to signal MPAM monitor overflow.

If this field is 0, the MSC supports generating a hardwired interrupt for monitor overflow events.

If this field is 0 and the HAS_OFLW_MSI field in this register is 1, the MSC supports generating both hardwired interrupts and MSI writes to signal interrupts.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_OFLW_MSI, bit [29]
When FEAT_MPAMv1p1 is implemented:

Has support for MSI writes to signal MPAM monitor overflow interrupts. These registers are implemented: MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA and MSMON_OFLOW_MSI_MPAM.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_OFLW_MSI	Meaning
0b0	MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA and MSMON_OFLOW_MSI_MPAM registers are not implemented.
0b1	MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA and MSMON_OFLOW_MSI_ATTR are implemented and can be used to generate writes to signal MPAM monitor overflow interrupts.

If MPAMF_MSMON_IDR.NO_HW_OFLW_INTR is 1 and this bit is 0, this MSC does not support monitor overflow interrupts.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_OFLOW_SR, bit [28]
When FEAT_MPAMv1p1 is implemented:

Has MPAM monitor overflow status register MSMON_OFLOW_SR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_OFLOW_SR	Meaning
0b0	Does not have MSMON_OFLOW_SR.
0b1	Supports MSMON_OFLOW_SR.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HAS_TL_MONITORING, bits [27:26]

When FEAT_MPAM_MSC_DOMAINS is implemented:

Indicates whether the monitor supports filtering based on ingress unstranslated PARTIDs, translated egress PARTIDs, or intermediate request PARTIDs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_TL_MONITORING	Meaning	Applies when
0b00	The monitor filters based on intermediate request PARTID: that is the reqPARTID in case of monitors in MSCs or the translated PARTID after ingress translation or before egress translation.	
0b01	The monitor filters based on incoming untranslated request PARTID.	When MPAMF_IDR.HAS_IN_TL == 1
0b10	The monitor filters based on outgoing translated request PARTID.	When MPAMF_IDR.HAS_OUT_TL == 1
0b11	Reserved.	

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [25:18]

Reserved, RES0.

MSMON_MBWU, bit [17]

Memory bandwidth usage monitoring. Indicates whether MPAM monitoring for Memory Bandwidth Usage by PARTID and PMG is implemented and whether the following bandwidth usage registers are accessible:

- MPAMF_MBWUMON_IDR, MSMON_CFG_MBWU_CTL, MSMON_CFG_MBWU_FLT, MSMON_MBWU.
- The optional MSMON_MBWU_CAPTURE.
- If MPAM v0.1 or MPAM v1.1 is implemented, the optional MSMON_MBWU_L and the optional MSMON_MBWU_L_CAPTURE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MSMON_MBWU	Meaning
0b0	Does not have monitoring for memory bandwidth usage and does not use the bandwidth usage registers.
0b1	Has monitoring of memory bandwidth usage and uses the bandwidth usage registers.

If RIS is implemented, this field indicates that memory bandwidth usage monitoring is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS as described in MPAMF_MBWUMON_IDR.

Access to this field is **RO**.

MSMON_CSU, bit [16]

Cache storage usage monitoring. Indicates whether MPAM monitoring of cache storage usage by PARTID and PMG is implemented and the following registers are accessible:

- MPAMF_CSUMON_IDR, MSMON_CFG_CSU_CTL, MSMON_CFG_CSU_FLT, MSMON_CSU.
- The optional MSMON_CSU_CAPTURE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MSMON_CSU	Meaning
0b0	Does not have monitoring for cache storage usage or the MPAMF_CSUMON_IDR, MSMON_CFG_CSU_CTL, MSMON_CFG_CSU_FLT, MSMON_CSU or MSMON_CSU_CAPTURE registers.
0b1	Has monitoring of cache storage usage and the MPAMF_CSUMON_IDR, MSMON_CFG_CSU_CTL, MSMON_CFG_CSU_FLT, MSMON_CSU and optional MSMON_CSU_CAPTURE registers.

If RIS is implemented, this field indicates that cache storage usage monitoring is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS as described in MPAMF_CSUMON_IDR.

Access to this field is **RO**.

Bits [15:0]

Reserved, RES0.

Accessing MPAMF_MSMON_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_MSMON_IDR is read-only.

MPAMF_MSMON_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_MSMON_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_MSMON_IDR_s is permitted to have either the same or different contents to MPAMF_MSMON_IDR_ns, MPAMF_MSMON_IDR_rt, or MPAMF_MSMON_IDR_rl.
- MPAMF_MSMON_IDR_ns is permitted to have either the same or different contents to MPAMF_MSMON_IDR_rt or MPAMF_MSMON_IDR_rl.
- MPAMF_MSMON_IDR_rt is permitted to have either the same or different contents to MPAMF_MSMON_IDR_rl.

There must be separate registers in the Secure (MPAMF_MSMON_IDR_s), Non-secure (MPAMF_MSMON_IDR_ns), Root (MPAMF_MSMON_IDR_rt), and Realm (MPAMF_MSMON_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_MSMON_IDR shows the configuration of memory system monitoring for the resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

Access to MPAMF_MSMON_IDR is not affected by MSMON_CFG_MON_SEL.RIS.

MPAMF_MSMON_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0080	MPAMF_MSMON_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0080	MPAMF_MSMON_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0080	MPAMF_MSMON_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0080	MPAMF_MSMON_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_OUT_TL_IDR, MPAM Egress PARTID Translation ID Register

The MPAMF_OUT_TL_IDR characteristics are:

Purpose

Indicates the egress PARTID translation capabilities of the MSC.

Configuration

The power domain of MPAMF_OUT_TL_IDR is IMPLEMENTATION DEFINED.

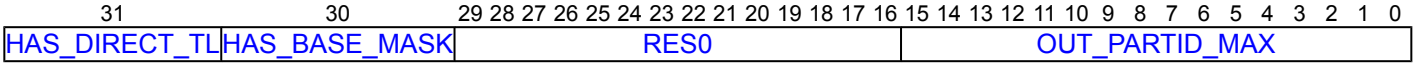
This register is present only when FEAT_MPAM_MSC_DOMAINS is implemented and MPAMF_IDR.HAS_OUT_TL == 1. Otherwise, direct accesses to MPAMF_OUT_TL_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_OUT_TL_IDR is a 32-bit register.

Field descriptions



HAS_DIRECT_TL, bit [31]

Indicates support for direct egress translation of PARTIDs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_DIRECT_TL	Meaning
0b0	Direct egress translation of PARTIDs is not supported.
0b1	Direct egress translation of PARTIDs is supported for those PARTIDs with an explicitly set translation configuration.

Access to this field is **RO**.

HAS_BASE_MASK, bit [30]

Indicates support for computed egress translation of PARTIDs using a configurable mask and base.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_BASE_MASK	Meaning
0b0	Computed egress translation of PARTIDs using a configurable mask and base is not supported.
0b1	Computed egress translation of PARTIDs using a configurable mask and base is supported for those PARTIDs without an explicitly set translation configuration.

Access to this field is **RO**.

Bits [29:16]

Reserved, RES0.

OUT_PARTID_MAX, bits [15:0]

When MPAMF_OUT_TL_IDR.HAS_DIRECT_TL == 1:

Maximum value for PARTIDs to be used as direct egress translations, as configured in MPAMCFG_OUT_TL.PARTID_TL.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing MPAMF_OUT_TL_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_OUT_TL_IDR is read-only.

MPAMF_OUT_TL_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_OUT_TL_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_OUT_TL_IDR_s is permitted to have either the same or different contents to MPAMF_OUT_TL_IDR_ns, MPAMF_OUT_TL_IDR_rt, or MPAMF_OUT_TL_IDR_rl.
- MPAMF_OUT_TL_IDR_ns is permitted to have either the same or different contents to MPAMF_OUT_TL_IDR_rt or MPAMF_OUT_TL_IDR_rl.
- MPAMF_OUT_TL_IDR_rt is permitted to have either the same or different contents to MPAMF_OUT_TL_IDR_rl.

There must be separate registers in the Secure (MPAMF_OUT_TL_IDR_s), Non-secure (MPAMF_OUT_TL_IDR_ns), Root (MPAMF_OUT_TL_IDR_rt), and Realm (MPAMF_OUT_TL_IDR_rl) MPAM feature pages.

MPAMF_OUT_TL_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x3200	MPAMF_OUT_TL_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x3200	MPAMF_OUT_TL_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x3200	MPAMF_OUT_TL_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x3200	MPAMF_OUT_TL_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_PARTID_NRW_IDR, MPAM PARTID Narrowing ID register

The MPAMF_PARTID_NRW_IDR characteristics are:

Purpose

Indicates the largest internal PARTID for this MSC.

MPAMF_PARTID_NRW_IDR_s indicates the largest Secure internal PARTID. MPAMF_PARTID_NRW_IDR_ns indicates the largest Non-secure internal PARTID.

When FEAT_RME is implemented: MPAMF_PARTID_NRW_rt indicates the largest Root internal PARTID. MPAMF_PARTID_NRW_rl indicates the largest Realm internal PARTID.

PARTID narrowing is global to the MSC and does not vary by resource instance.

Configuration

The power domain of MPAMF_PARTID_NRW_IDR is IMPLEMENTATION DEFINED.

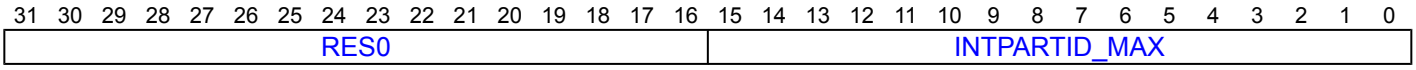
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PARTID_NRW == 1. Otherwise, direct accesses to MPAMF_PARTID_NRW_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_PARTID_NRW_IDR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

INTPARTID_MAX, bits [15:0]

The largest intPARTID supported in this MSC.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPAMF_PARTID_NRW_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_PARTID_NRW_IDR is read-only.

MPAMF_PARTID_NRW_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_PARTID_NRW_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_PARTID_NRW_IDR_s is permitted to have either the same or different contents to MPAMF_PARTID_NRW_IDR_ns, MPAMF_PARTID_NRW_IDR_rt, or MPAMF_PARTID_NRW_IDR_rl.
- MPAMF_PARTID_NRW_IDR_ns is permitted to have either the same or different contents to MPAMF_PARTID_NRW_IDR_rt or MPAMF_PARTID_NRW_IDR_rl.
- MPAMF_PARTID_NRW_IDR_rt is permitted to have either the same or different contents to MPAMF_PARTID_NRW_IDR_rl.

There must be separate registers in the Secure (MPAMF_PARTID_NRW_IDR_s), Non-secure (MPAMF_PARTID_NRW_IDR_ns), Root (MPAMF_PARTID_NRW_IDR_rt), and Realm (MPAMF_PARTID_NRW_IDR_rl) MPAM feature pages.

MPAMF_PARTID_NRW_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0050	MPAMF_PARTID_NRW_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0050	MPAMF_PARTID_NRW_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0050	MPAMF_PARTID_NRW_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0050	MPAMF_PARTID_NRW_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_PRI_IDR, MPAM Priority Partitioning Identification Register

The MPAMF_PRI_IDR characteristics are:

Purpose

Indicates which MPAM priority partitioning features are present on this MSC.

MPAMF_PRI_IDR_s indicates priority partitioning features accessed from the Secure MPAM feature page. MPAMF_PRI_IDR_ns indicates priority partitioning features accessed from the Non-secure MPAM feature page. MPAMF_PRI_IDR_rt indicates priority partitioning features accessed from the Root MPAM feature page. MPAMF_PRI_IDR_rl indicates priority partitioning features accessed from the Realm MPAM feature page.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selected by MPAMCFG_PART_SEL.RIS. The description of every field that is affected by MPAMCFG_PART_SEL.RIS has that information within the field description.

Configuration

The power domain of MPAMF_PRI_IDR is IMPLEMENTATION DEFINED.

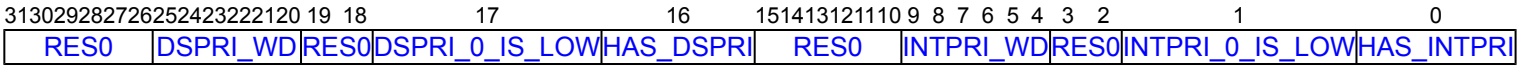
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PRI_PART == 1. Otherwise, direct accesses to MPAMF_PRI_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_PRI_IDR is a 32-bit register.

Field descriptions



Bits [31:26]

Reserved, RES0.

DSPRI_WD, bits [25:20]

Number of implemented bits in the downstream priority field (DSPRI) of MPAMCFG_PRI.

If HAS_DSPRI == 1, this field must contain a value from 1 to 16, inclusive.

If HAS_DSPRI == 0, this field must be 0.

If RIS is implemented, this field indicates the number of downstream priority bits for the resource instance selected by MPAMCFG_PART_SEL.RIS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [19:18]

Reserved, RES0.

DSPRI_0_IS_LOW, bit [17]

Indicates whether 0 in MPAMCFG_PRI.DSPRI is the lowest or the highest downstream priority.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DSPRI_0_IS_LOW	Meaning
0b0	In the MPAMCFG_PRI.DSPRI field, a value of 0 means the highest priority.
0b1	In the MPAMCFG_PRI.DSPRI field, a value of 0 means the lowest priority.

If RIS is implemented, this field indicates that 0 is the lowest downstream priority for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

HAS_DSPRI, bit [16]

Indicates that the MPAMCFG_PRI register implements the DSPRI field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_DSPRI	Meaning
0b0	This MSC supports priority partitioning, but does not implement a downstream priority (DSPRI) field in the MPAMCFG_PRI register.
0b1	This MSC supports downstream priority partitioning and implements the downstream priority (DSPRI) field in the MPAMCFG_PRI register.

If RIS is implemented, this field indicates that downstream priority is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Bits [15:10]

Reserved, RES0.

INTPRI_WD, bits [9:4]

Number of implemented bits in the internal priority field (INTPRI) in the MPAMCFG_PRI register.

If HAS_INTPRI == 1, this field must contain a value from 1 to 16, inclusive.

If HAS_INTPRI == 0, this field must be 0.

If RIS is implemented, this field indicates the number of internal priority bits for the resource instance selected by MPAMCFG_PART_SEL.RIS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [3:2]

Reserved, RES0.

INTPRI_0_IS_LOW, bit [1]

Indicates whether 0 in MPAMCFG_PRI.INTPRI is the lowest or the highest internal priority.

The value of this field is an IMPLEMENTATION DEFINED choice of:

INTPRI_0_IS_LOW	Meaning
0b0	In the MPAMCFG_PRI.INTPRI field, a value of 0 means the highest priority.
0b1	In the MPAMCFG_PRI.INTPRI field, a value of 0 means the lowest priority.

If RIS is implemented, this field indicates that 0 is the lowest internal priority for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

HAS_INTPRI, bit [0]

Indicates that this MSC implements the INTPRI field in the MPAMCFG_PRI register.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_INTPRI	Meaning
0b0	This MSC supports priority partitioning, but does not implement the internal priority (INTPRI) field in the MPAMCFG_PRI register.
0b1	This MSC supports internal priority partitioning and implements the internal priority (INTPRI) field in the MPAMCFG_PRI register.

If RIS is implemented, this field indicates that internal priority is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Access to this field is **RO**.

Accessing MPAMF_PRI_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_PRI_IDR is read-only.

MPAMF_PRI_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_PRI_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_PRI_IDR_s is permitted to have either the same or different contents to MPAMF_PRI_IDR_ns, MPAMF_PRI_IDR_rt, or MPAMF_PRI_IDR_rl.
- MPAMF_PRI_IDR_ns is permitted to have either the same or different contents to MPAMF_PRI_IDR_rt or MPAMF_PRI_IDR_rl.
- MPAMF_PRI_IDR_rt is permitted to have either the same or different contents to MPAMF_PRI_IDR_rl.

There must be separate registers in the Secure (MPAMF_PRI_IDR_s), Non-secure (MPAMF_PRI_IDR_ns), Root (MPAMF_PRI_IDR_rt), and Realm (MPAMF_PRI_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_PRI_IDR shows the configuration of priority partitioning for the resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_PRI_IDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0048	MPAMF_PRI_IDR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0048	MPAMF_PRI_IDR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0048	MPAMF_PRI_IDR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0048	MPAMF_PRI_IDR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_SIDR, MPAM Features Secure Identification Register

The MPAMF_SIDR characteristics are:

Purpose

The MPAMF_SIDR is a 32-bit read-only register that indicates the maximum Secure PARTID and Secure PMG on this MSC.

Configuration

The power domain of MPAMF_SIDR is IMPLEMENTATION DEFINED.

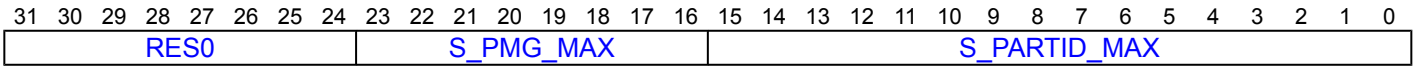
This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_SIDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_SIDR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

S_PMG_MAX, bits [23:16]

Maximum value of Secure PMG supported by this component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

S_PARTID_MAX, bits [15:0]

Maximum value of Secure PARTID supported by this component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MPAMF_SIDR

This register is only within the Secure MPAM feature page memory frame.

MPAMF_SIDR is read-only.

MPAMF_SIDR must only be readable from the Secure MPAM feature page. If the system or the MSC does not support the Secure address map, this register must not be accessible.

MPAMF_SIDR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0008	MPAMF_SIDR_s

Accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CAPT_EVNT, MPAM Capture Event Generation Register

The MSMON_CAPT_EVNT characteristics are:

Purpose

Generates a local capture event when written with bit[0] as 1.

MSMON_CAPT_EVNT_s generates local capture events for Secure monitor instances only or for Secure and Non-secure monitor instances. MSMON_CAPT_EVNT_ns generates local capture events for Non-secure monitor instances only. MSMON_CAPT_EVNT_rt generates local capture events for Root monitor instances only or for Root, Secure, Realm, and Non-secure monitor instances. MSMON_CAPT_EVNT_rl generates local capture events for Realm monitor instances or for for Realm monitor instances or Realm and Non-secure monitor instances.

Configuration

The power domain of MSMON_CAPT_EVNT is IMPLEMENTATION DEFINED.

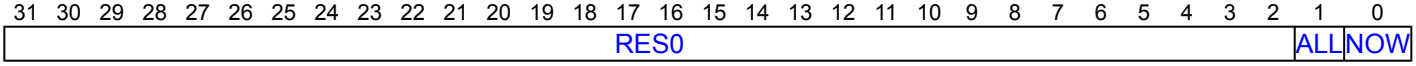
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, and MPAMF_MSMON_IDR.HAS_LOCAL_CAPT_EVNT == 1. Otherwise, direct accesses to MSMON_CAPT_EVNT are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CAPT_EVNT is a 32-bit register.

Field descriptions



Bits [31:2]

Reserved, RES0.

ALL, bit [1]

In the Secure instance of this register:

- If ALL is written as 1 and NOW is also written as 1, signal a capture event to Secure and Non-secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.
- If ALL is written as 0 and NOW is written as 1, signal a capture event to Secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.

In the Non-secure instance of this register, this field is RAZ/WI.

In the Root instance of this register:

- If ALL is written as 1 and NOW is also written as 1, signal a capture event to Root, Realm, Secure, and Non-secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.
- If ALL is written as 0 and NOW is written as 1, signal a capture event to Root monitor instances within this MSC that are configured with CAPT_EVNT = 7.

In the Realm instance of this register:

- If ALL is written as 1 and NOW is also written as 1, signal a capture event to Realm and Non-secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.
- If ALL is written as 0 and NOW is written as 1, signal a capture event to Realm monitor instances within this MSC that are configured with CAPT_EVNT = 7.

This bit always reads as zero.

ALL	Meaning
0b0	Send capture event only to monitor instances in the same MPAM feature page as this register.
0b1	Send capture event to monitor instances in certain MPAM feature pages as described in the ALL field of this register.

NOW, bit [0]

When written as 1, this bit causes an event to those monitor instances described in the ALL field that have CAPT_EVNT set to the value of 7.

When this bit is written as 0, no event is signaled.

This bit always reads as zero.

Accessing MSMON_CAPT_EVNT

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CAPT_EVNT_s must only be accessible from the Secure MPAM feature page. MSMON_CAPT_EVNT_ns must only be accessible from the Non-secure MPAM feature page.

MSMON_CAPT_EVNT_s and MSMON_CAPT_EVNT_ns must be separate registers. The Secure instance (MSMON_CAPT_EVNT_s) can generate local capture events for Secure monitor instances only or for Secure and Non-secure monitor instances, and the Non-secure instance (MSMON_CAPT_EVNT_ns) can generate local capture events for Non-secure monitor instances only.

MSMON_CAPT_EVNT can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0808	MSMON_CAPT_EVNT_s

Accesses to this register are **WO/RAZ**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0808	MSMON_CAPT_EVNT_ns

Accesses to this register are **WO/RAZ**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0808	MSMON_CAPT_EVNT_rt

When FEAT_RME is implemented, accesses to this register are **WO/RAZ**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0808	MSMON_CAPT_EVNT_rl

When FEAT_RME is implemented, accesses to this register are **WO/RAZ**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CFG_CSU_CTL, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register

The MSMON_CFG_CSU_CTL characteristics are:

Purpose

Controls the CSU monitor selected by MSMON_CFG_MON_SEL.

MSMON_CFG_CSU_CTL_s controls the Secure cache storage usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL_ns controls Non-secure cache storage usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL_rt controls the monitor configuration for the Root PARTID selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL_rl controls the monitor configuration for the Realm PARTID selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance configuration accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configuration

The power domain of MSMON_CFG_CSU_CTL is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_CTL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_CSU_CTL is a 32-bit register.

Field descriptions

EN, bit [31]

Enabled.

EN	Meaning
0b0	The monitor instance is disabled and must not collect any information.
0b1	The monitor instance is enabled to collect information according to the configuration of the instance.

CAPT_EVNT, bits [30:28]

Capture event selector.

Select the event that triggers capture from the following:

CAPT_EVNT	Meaning
0b000	No capture event is triggered.
0b001	External capture event 1 (optional, but recommended)
0b010	External capture event 2 (optional)
0b011	External capture event 3 (optional)
0b100	External capture event 4 (optional)
0b101	External capture event 5 (optional)
0b110	External capture event 6 (optional)
0b111	Capture occurs when a MSMON_CAPT_EVNT register in this MSC is written and causes a capture event for the Security state of this monitor. (optional)

The values marked as optional indicate capture event sources that can be omitted in an implementation. Those values representing non-implemented event sources must not trigger a capture event.

When MPAMF_CSUMON_IDR.HAS_CAPTURE == 0, access to this field is **RAZ/WI**.

CAPT_RESET, bit [27]

Reset after capture.

Controls whether the value of MSMON_CSU is reset to zero immediately after being copied to MSMON_CSU_CAPTURE.

CAPT_RESET	Meaning
0b0	Monitor is not reset on capture.
0b1	Monitor is reset on capture.

Because the CSU monitor type produces a measurement rather than a count, it might not make sense to ever reset the value after a capture. If there is no reason to ever reset a CSU monitor, this field is RAZ/WI.

When MPAMF_CSUMON_IDR.HAS_CAPTURE == 0, access to this field is **RAZ/WI**.

OFLOW_STATUS, bit [26]

Overflow status.

Indicates whether the value of MSMON_CSU has overflowed.

If MPAMF_CSUMON_IDR.HAS_CEVNT_OFLW is 1 or MPAMF_CSUMON_IDR.HAS_OFLOW_LNKG is 1, then a store to MSMON_CSU when this field is 1 resets this field to 0.

OFLOW_STATUS	Meaning
0b0	No overflow has occurred.
0b1	At least one overflow has occurred since this bit was last written to zero.

If overflow is not possible for a CSU monitor in the implementation, this field is RAZ/WI.

OFLOW_INTR, bit [25]

Overflow Interrupt.

Controls whether an overflow interrupt is generated when the value of MSMON_CSU has overflowed.

OFLOW_INTR	Meaning
0b0	No interrupt is signaled on an overflow of MSMON_CSU.
0b1	On overflow, an implementation-specific interrupt is signaled.

When MSMON_CFG_CSU_CTL.OFLOW_INTR == 0, access to this field is **RAZ/WI**.

OFLOW_FRZ, bit [24]

Freeze Monitor on Overflow.

Controls whether the value of MSMON_CSU.VALUE freezes on an overflow.

OFLOW_FRZ	Meaning
0b0	Monitor count wraps on overflow.
0b1	Monitor count freezes on overflow. The frozen value might be 0 or another value if the monitor overflowed with an increment larger than 1.

If overflow is not possible for a CSU monitor in the implementation, this field is RAZ/WI.

When a MSMON_CSU.VALUE of a monitor instance is frozen it does not change until MSMON_CSU register for that instance has been written.

OFLOW_CAPT, bit [23]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_CSUMON_IDR.HAS_OFLOW_CAPT == 1:

Capture Monitor on Overflow.

OFLOW_CAPT	Meaning
0b0	Monitor is not captured on an overflow or when affected by an overflow linkage event.
0b1	Monitor is captured and the MSMON_CSU.{NRDY, VALUE} fields are copied to the monitor instance's capture register on an overflow or when affected by an overflow linkage event. The monitor instance treats an overflow of this monitor instance as a private capture event. If MSMON_CFG_MBWU_CTL.CEVNT_OFLW is 1, this monitor instance also treats an overflow linkage event as a capture event. If the OFLOW_FRZ field is 1, the monitor does not continue to count after the overflow or overflow linkage event. If the CAPT_RESET field is 1, the monitor instance resets to 0.

Otherwise:

Reserved, RES0.

SUBTYPE, bits [22:20]

Subtype. Type of cache storage usage counted by this monitor.

This field is not currently used for CSU monitors, but reserved for future use.

This field is RAZ/WI.

Bit [19]

Reserved, RES0.

CEVNT_OFLW, bit [18]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_CSUMON_IDR.HAS_CEVNT_OFLW == 1:

Capture Event performs overflow behavior.

CEVNT_OFLW	Meaning
0b0	On a capture event matching the CAPT_EVNT field, the capture behaviors are performed. The MSMON_CSU.{VALUE, NRDY} fields are transferred to the monitor instance's capture register.
0b1	On a capture event matching the CAPT_EVNT field, the monitor instance treats a capture event as an overflow and the overflow behaviors are performed. The behavior is controlled by the MSMON_CFG_CSU_CTL.{OFLOW_FRZ, OFLOW_CAPT, CAPT_RESET} fields. The MSMON_CFG_CSU_CTL.OFLOW_STATUS field is set for this monitor instance.

Otherwise:

Reserved, RES0.

MATCH_PMG, bit [17]

Match PMG.

Controls whether the monitor measures only storage used with PMG matching MSMON_CFG_CSU_FLT.PMG.

MATCH_PMG	Meaning
0b0	The monitor measures storage used with any PMG value.
0b1	The monitor only measures storage used with the PMG value matching MSMON_CFG_CSU_FLT.PMG.

If MATCH_PMG is 1 and MATCH_PARTID is 0, it is CONSTRAINED UNPREDICTABLE whether the monitor instance:

- Measures the storage used with matching PMG and with any PARTID.
- Measures no storage usage, that is, MSMON_CSU.VALUE is zero.
- Measures the storage used with matching PMG and PARTID, that is, treats MATCH_PARTID as == 1.

MATCH_PARTID, bit [16]

Match PARTID.

Controls whether the monitor measures only storage used with PARTID matching MSMON_CFG_CSU_FLT.PARTID.

MATCH_PARTID	Meaning
0b0	The monitor measures storage used with any PARTID value.
0b1	The monitor only measures storage used with the PARTID value matching MSMON_CFG_CSU_FLT.PARTID.

Bits [15:11]

Reserved, RES0.

OFLOW_LNKG, bits [10:8]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_CSUMON_IDR.HAS_OFLOW_LNKG == 1:

Overflow linkage event.

Controls signaling of a capture event on overflow of this monitor instance.

OFLOW_LNKG	Meaning
0b000	Overflow of the monitor instance only affects this monitor instance.
0b001	Overflow of this monitor instance signals Capture Event 1.
0b010	Overflow of this monitor instance signals Capture Event 2.
0b011	Overflow of this monitor instance signals Capture Event 3.
0b100	Overflow of this monitor instance signals Capture Event 4.
0b101	Overflow of this monitor instance signals Capture Event 5.
0b110	Overflow of this monitor instance signals Capture Event 6.
0b111	Reserved.

Otherwise:

Reserved, RES0.

TYPE, bits [7:0]

Monitor Type Code. The CSU monitor is TYPE = 0x43.

TYPE is a read-only constant indicating the type of the monitor.

Reads as 0x43.

Access to this field is **RO**.

Accessing MSMON_CFG_CSU_CTL

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CFG_CSU_CTL_s must only be accessible from the Secure MPAM feature page.
- MSMON_CFG_CSU_CTL_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_CFG_CSU_CTL_rt must only be accessible from the Root MPAM feature page.
- MSMON_CFG_CSU_CTL_rl must only be accessible from the Realm MPAM feature page.

MSMON_CFG_CSU_CTL_s, MSMON_CFG_CSU_CTL_ns, MSMON_CFG_CSU_CTL_rt, and MSMON_CFG_CSU_CTL_rl must be separate registers:

- The Secure instance (MSMON_CFG_CSU_CTL_s) accesses the cache storage usage monitor controls used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_CSU_CTL_ns) accesses the cache storage usage monitor controls used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_CSU_CTL_rt) accesses the cache storage usage monitor controls used for Root PARTIDs.
- The Realm instance (MSMON_CFG_CSU_CTL_rl) accesses the cache storage usage monitor controls used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_CSU_CTL access the cache storage usage monitor configuration settings for the cache resource instance selected by MSMON_CFG_MON_SEL.RIS and the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_CSU_CTL access the cache storage usage monitor configuration settings for the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_CFG_CSU_CTL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0818	MSMON_CFG_CSU_CTL_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0818	MSMON_CFG_CSU_CTL_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0818	MSMON_CFG_CSU_CTL_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0818	MSMON_CFG_CSU_CTL_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CFG_CSU_FLT, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register

The MSMON_CFG_CSU_FLT characteristics are:

Purpose

Configures PARTID and PMG to measure or count in the CSU monitor selected by MSMON_CFG_MON_SEL.

MSMON_CFG_CSU_FLT_s sets filter conditions for the Secure cache storage usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL_ns sets filter conditions for the Non-secure cache storage usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_FLT_rt sets the filter conditions for the Root PARTID selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_FLT_rl sets the filter conditions for the Realm PARTID selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance filter configuration accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configuration

The power domain of MSMON_CFG_CSU_FLT is IMPLEMENTATION DEFINED.

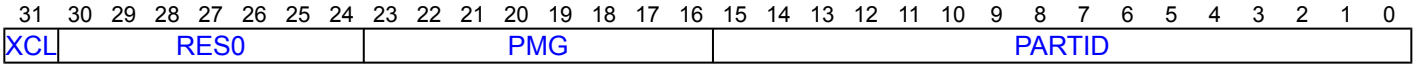
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_FLT are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_CSU_FLT is a 32-bit register.

Field descriptions



XCL, bit [31]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_CSUMON_IDR.HAS_XCL == 1:

Exclude Clean. The monitor instance does not count cache storage used by lines in an unmodified cache state.

XCL	Meaning
0b0	Monitor instance counts cache storage in modified and unmodified cache lines.
0b1	Monitor instance counts cache storage in modified cache lines only.

Otherwise:

Reserved, RES0.

Bits [30:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group to filter cache storage usage monitoring.

If MSMON_CFG_CSU_CTL.MATCH_PMG is 0, this field is not used to match cache storage to a PMG and the contents of this field is ignored.

If MSMON_CFG_CSU_CTL.MATCH_PMG is 1 and MSMON_CFG_CSU_CTL.MATCH_PARTID is 1, the monitor instance selected by MSMON_CFG_MON_SEL measures or counts cache storage labeled with PMG equal to this field and PARTID equal to the PARTID field.

If MSMON_CFG_CSU_CTL.MATCH_PMG is 1 and MSMON_CFG_CSU_CTL.MATCH_PARTID is 0, the behavior of the monitor instance selected by MSMON_CFG_MON_SEL is CONSTRAINED UNPREDICTABLE. See MSMON_CFG_CSU_CTL.MATCH_PMG for more information.

PARTID, bits [15:0]

Partition ID to filter cache storage usage monitoring.

If MSMON_CFG_CSU_CTL.MATCH_PARTID is 0 and MSMON_CFG_CSU_CTL.MATCH_PMG is 0, the monitor measures all allocated cache storage.

If MSMON_CFG_CSU_CTL.MATCH_PARTID is 0 and MSMON_CFG_CSU_CTL.MATCH_PMG is 1, the behavior of the monitor is CONSTRAINED UNPREDICTABLE. See the description of MSMON_CFG_CSU_CTL.MATCH_PMG.

If MSMON_CFG_CSU_CTL.MATCH_PARTID is 1 and MSMON_CFG_CSU_CTL.MATCH_PMG is 0, the monitor selected by MSMON_CFG_MON_SEL measures or counts cache storage labeled with PARTID equal to this field.

If MSMON_CFG_CSU_CTL.MATCH_PARTID is 1 and MSMON_CFG_CSU_CTL.MATCH_PMG is 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts cache storage labeled with PARTID equal to this field and PMG equal to the PMG field.

Accessing MSMON_CFG_CSU_FLT

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CFG_CSU_FLT_s must only be accessible from the Secure MPAM feature page.
- MSMON_CFG_CSU_FLT_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_CFG_CSU_FLT_rt must only be accessible from the Root MPAM feature page.
- MSMON_CFG_CSU_FLT_rl must only be accessible from the Realm MPAM feature page.

MSMON_CFG_CSU_FLT_s, MSMON_CFG_CSU_FLT_ns, MSMON_CFG_CSU_FLT_rt, and MSMON_CFG_CSU_FLT_rl must be separate registers:

- The Secure instance (MSMON_CFG_CSU_FLT_s) accesses the PARTID and PMG matching for a cache storage usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_CSU_FLT_ns) accesses the PARTID and PMG matching for a cache storage usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_CSU_FLT_rt) accesses the PARTID and PMG matching for a cache storage usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_CFG_CSU_FLT_rl) accesses the PARTID and PMG matching for a cache storage usage monitor used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_CSU_FLT access the monitor configuration settings for the resource instance selected by MSMON_CFG_MON_SEL.RIS and the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_CSU_FLT access the monitor configuration settings for the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_CFG_CSU_FLT can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0810	MSMON_CFG_CSU_FLT_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance

MPAM	MPAMF_BASE_ns	0x0810	MSMON_CFG_CSU_FLT_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0810	MSMON_CFG_CSU_FLT_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0810	MSMON_CFG_CSU_FLT_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CFG_MBWU_CTL, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register

The MSMON_CFG_MBWU_CTL characteristics are:

Purpose

Controls the MBWU monitor selected by MSMON_CFG_MON_SEL.

MSMON_CFG_MBWU_CTL_s controls the Secure memory bandwidth usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_MBWU_CTL_ns controls Non-secure memory bandwidth usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_MBWU_CTL_rt controls the monitor configuration for the Root PARTID selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CFG_MBWU_CTL_rl controls the monitor configuration for the Realm PARTID selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance configuration accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configuration

The power domain of MSMON_CFG_MBWU_CTL is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_CFG_MBWU_CTL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_MBWU_CTL is a 32-bit register.

Field descriptions

EN, bit [31]

Enabled.

EN	Meaning
0b0	The monitor instance is disabled and must not collect any information.
0b1	The monitor instance is enabled to collect information according to the configuration of the instance.

CAPT_EVNT, bits [30:28]

Capture event selector.

When the selected capture event occurs, MSMON_MBWU of the monitor instance is copied to MSMON_MBWU_CAPTURE of the same instance. If the long counter is also implemented, MSMON_MBWU_L is also copied to MSMON_MBWU_L_CAPTURE.

Select the event that triggers capture from the following:

CAPT_EVNT	Meaning
0b000	No capture event is triggered.
0b001	External capture event 1 (optional, but recommended)
0b010	External capture event 2 (optional)
0b011	External capture event 3 (optional)
0b100	External capture event 4 (optional)
0b101	External capture event 5 (optional)
0b110	External capture event 6 (optional)
0b111	Capture occurs when a MSMON_CAPT_EVNT register in this MSC is written and causes a capture event for the Security state of this monitor. (optional)

The values marked as optional indicate capture event sources that can be omitted in an implementation. Those values representing non-implemented event sources must not trigger a capture event.

When MPAMF_MBWUMON_IDR.HAS_CAPTURE == 0, access to this field is **RAZ/WI**.

CAPT_RESET, bit [27]

Reset MSMON_MBWU.VALUE after capture.

Controls whether the VALUE field of the monitor instance is reset to zero immediately after being copied to the corresponding capture register.

CAPT_RESET	Meaning
0b0	MSMON_MBWU.VALUE field of the monitor instance is not reset on capture.
0b1	MSMON_MBWU.VALUE field of the monitor instance is reset on capture.

This control bit affects both MSMON_MBWU and MSMON_MBWU_L in implementations that include MSMON_MBWU_L.

When MPAMF_MBWUMON_IDR.HAS_CAPTURE == 0, access to this field is **RAZ/WI**.

OFLOW_STATUS, bit [26]

Overflow status.

Indicates whether the value of MSMON_MBWU has overflowed.

OFLOW_STATUS	Meaning
0b0	MSMON_MBWU.VALUE has not overflowed.
0b1	MSMON_MBWU.VALUE has overflowed at least once since this bit was last written to zero.

Overflow status for MSMON_MBWU_L.VALUE is reported in MSMON_CFG_MBWU_CTL.OFLOW_STATUS_L.

If MPAMF_MBWUMON_IDR.HAS_CEVNT_OFLW is 1 or MPAMF_MBWUMON_IDR.HAS_OFLOW_LNKG is 1, then a store to MSMON_MBWU when this field is 1 resets this field to 0.

OFLOW_INTR, bit [25]

Enable interrupt on overflow of MSMON_MBWU.VALUE.

OFLOW_INTR	Meaning
0b0	No interrupt is signaled on an overflow of MSMON_MBWU.VALUE.
0b1	An implementation-specific interrupt is signaled on an overflow of MSMON_MBWU.VALUE.

Interrupt enable for overflow of MSMON_MBWU_L.VALUE is controlled by MSMON_CFG_MBWU_CTL.OFLOW_INTR_L.

When MSMON_CFG_MBWU_CTL.OFLOW_INTR == 0, access to this field is **RAZ/WI**.

OFLOW_FRZ, bit [24]

Freeze monitor instance on overflow.

Controls whether MSMON_MBWU.VALUE field of the monitor instance freezes on an overflow.

OFLOW_FRZ	Meaning
0b0	MSMON_MBWU.VALUE field of the monitor instance wraps on overflow.
0b1	MSMON_MBWU.VALUE field of the monitor instance freezes on overflow. If the increment that caused the overflow was 1, the frozen value is the post-increment value of 0. If the increment that caused the overflow was larger than 1, the frozen value of the monitor might be 0 or a larger value less than the final increment.

When a MSMON_MBWU.VALUE of a monitor instance is frozen it does not change until MSMON_CSU register for that instance has been written. If the monitor implements both MSMON_MBWU and MSMON_MBWU_L registers, both are frozen. A write to a frozen register unfreezes the count for just that register.

This control bit affects both MSMON_MBWU and MSMON_MBWU_L in implementations that include MSMON_MBWU_L.

OFLOW_CAPT, bit [23]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_MBWUMON_IDR.HAS_OFLOW_CAPT == 1:

Capture Monitor on Overflow.

OFLOW_CAPT	Meaning
0b0	Monitor register MSMON_MBWU is not captured on an overflow or when affected by an overflow linkage event.
0b1	Monitor register MSMON_MBWU is captured and the MSMON_MBWU.{NRDY, VALUE} fields are copied to the monitor instance's MSMON_MBWU_CAPTURE register on an overflow or when affected by an overflow linkage event. The monitor instance treats an overflow of this monitor instance as a private capture event. If MSMON_CFG_MBWU_CTL.CEVNT_OFLW is 1, this monitor instance also treats an overflow linkage event as a capture event. If OFLOW_FRZ is 1, the monitor does not continue to count after the overflow or overflow linkage event. If CAPT_RESET is 1, the monitor instance resets to 0.

This bit does not control whether MSMON_MBWU_L is captured on an overflow or overflow linkage event. See MSMON_CFG_MBWU_CTL.OFLOW_CAPT_L.

Otherwise:

Reserved, RES0.

SUBTYPE, bits [22:20]

Subtype. Type of bandwidth counted by this monitor.

This field is not currently used for MBWU monitors, but reserved for future use.

This field is RAZ/WI.

SCLEN, bit [19]

MSMON_MBWU.VALUE Scaling Enable.

Enables scaling of MSMON_MBWU.VALUE by MPAMF_MBWUMON_IDR.SCALE.

SCLEN	Meaning
0b0	MSMON_MBWU.VALUE has bytes counted by the monitor instance.
0b1	MSMON_MBWU.VALUE has bytes counted by the monitor instance, shifted right by MPAMF_MBWUMON_IDR.SCALE.

CEVNT_OFLW, bit [18]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_MBWUMON_IDR.HAS_CEVNT_OFLW == 1:

Capture Event performs overflow behavior.

CEVNT_OFLW	Meaning
0b0	On a capture event matching the CAPT_EVNT field the capture behaviors are performed. The NRDY and VALUE fields are transferred to the monitor instance's capture register.
0b1	On a capture event matching the CAPT_EVNT field the monitor instance treats a capture event as an overflow and the overflow behaviors are performed. The behavior is controlled by the MSMON_CFG_MBWU_CTL.{OFLOW_FRZ, OFLOW_CAPT, OFLOW_CAPT_L, CAPT_RESET} fields. The MSMON_CFG_MBWU_CTL.{OFLOW_STATUS, OFLOW_STATUS_L} fields are set for this monitor instance.

Otherwise:

Reserved, RES0.

MATCH_PMG, bit [17]

Match PMG.

Controls whether the monitor instance only counts data transferred with PMG matching MSMON_CFG_MBWU_FLT.PMG.

MATCH_PMG	Meaning
0b0	The monitor instance counts data transferred with any PMG value.
0b1	The monitor instance only counts data transferred with the PMG value matching MSMON_CFG_MBWU_FLT.PMG.

MATCH_PARTID, bit [16]

Match PARTID.

Controls whether the monitor instance counts only data transferred with PARTID matching MSMON_CFG_MBWU_FLT.PARTID.

MATCH_PARTID	Meaning
0b0	The monitor instance counts data transferred with any PARTID value.
0b1	The monitor instance only counts data transferred with the PARTID value matching MSMON_CFG_MBWU_FLT.PARTID.

OFLOW_STATUS_L, bit [15]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Overflow Status of MSMON_MBWU_L.VALUE of the monitor instance.

Indicates whether MSMON_MBWU_L.VALUE has overflowed.

OFLOW_STATUS_L	Meaning
0b0	MSMON_MBWU_L.VALUE has not overflowed.
0b1	MSMON_MBWU_L.VALUE has overflowed at least once since this bit was last written to zero.

If MPAMF_MBWUMON_IDR.HAS_LONG == 0, this bit is RES0.

Overflow status of MSMON_MBWU.VALUE is reported in MSMON_CFG_MBWU_CTL.OFLOW_STATUS.

If MPAMF_MBWUMON_IDR.HAS_CEVNT_OFLW is 1 or MPAMF_MBWUMON_IDR.HAS_OFLOW_LNKG is 1, then a store to MSMON_MBWU_L when this field is 1 resets this field to 0.

Otherwise:

Reserved, RES0.

OFLOW_INTR_L, bit [14]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_MBWUMON_IDR.HAS_LONG == 1:

Overflow Interrupt for MSMON_MBWU_L.

Controls whether an MPAM overflow interrupt is generated when MSMON_MBWU_L.VALUE overflows.

OFLOW_INTR_L	Meaning
0b0	No interrupt is signaled on an overflow of MSMON_MBWU_L.VALUE.
0b1	An implementation-specific interrupt is signaled on overflow of MSMON_MBWU_L.VALUE.

When MSMON_CFG_MBWU_CTL.OFLOW_INTR_L == 0, access to this field is **RAZ/WI**.

Otherwise:

Reserved, RES0.

OFLOW_CAPT_L, bit [13]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented), MPAMF_MBWUMON_IDR.HAS_LONG == 1, and MPAMF_MBWUMON_IDR.HAS_OFLOW_CAPT == 1:

Capture Long Monitor on Overflow.

Controls whether MSMON_MBWU_L is copied to MSMON_MBWU_L_CAPTURE on an overflow or an overflow linkage event.

OFLOW_CAPT_L	Meaning
0b0	Monitor register MSMON_MBWU_L is not captured on an overflow or when affected by an overflow linkage event.
0b1	Monitor register MSMON_MBWU_L is captured on an overflow or when affected by an overflow linkage event. If OFLOW_FRZ is 1, the monitor does not continue to count after the overflow or overflow linkage event. If CAPT_RESET is 1, the monitor instance resets to 0.

If this bit is 1, this monitor instance treats an overflow of this monitor instance as a private capture event.

If this bit is 1, this monitor instance also treats overflow linkage events for which it qualifies as a private capture event.

Otherwise:

Reserved, RES0.

Bits [12:11]

Reserved, RES0.

OFLOW_LNKG, bits [10:8]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_MBWUMON_IDR.HAS_OFLOW_LNKG == 1:

Overflow linkage event.

Controls signaling of a capture event on overflow of this monitor instance.

OFLOW_LNKG	Meaning
0b000	Overflow of the monitor instance only affects this monitor instance.
0b001	Overflow of this monitor instance signals Capture Event 1.
0b010	Overflow of this monitor instance signals Capture Event 2.
0b011	Overflow of this monitor instance signals Capture Event 3.
0b100	Overflow of this monitor instance signals Capture Event 4.
0b101	Overflow of this monitor instance signals Capture Event 5.
0b110	Overflow of this monitor instance signals Capture Event 6.
0b111	Reserved.

Otherwise:

Reserved, RES0.

TYPE, bits [7:0]

Monitor Type Code. The MBWU monitor is TYPE = 0x42.

TYPE is a read-only constant indicating the type of the monitor.

Reads as 0x42.

Access to this field is **RO**.

Accessing MSMON_CFG_MBWU_CTL

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CFG_MBWU_CTL_s must only be accessible from the Secure MPAM feature page.
- MSMON_CFG_MBWU_CTL_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_CFG_MBWU_CTL_rt must only be accessible from the Root MPAM feature page.
- MSMON_CFG_MBWU_CTL_rl must only be accessible from the Realm MPAM feature page.

MSMON_CFG_MBWU_CTL_s, MSMON_CFG_MBWU_CTL_ns, MSMON_CFG_MBWU_CTL_rt, and MSMON_CFG_MBWU_CTL_rl must be separate registers:

- The Secure instance (MSMON_CFG_MBWU_CTL_s) accesses the memory bandwidth usage monitor controls used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_MBWU_CTL_ns) accesses the memory bandwidth usage monitor controls used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_MBWU_CTL_rt) accesses the memory bandwidth usage monitor controls used for Root PARTIDs.
- The Realm instance (MSMON_CFG_MBWU_CTL_rl) accesses the memory bandwidth usage monitor controls used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_MBWU_CTL access the monitor configuration settings for the bandwidth resource instance selected by MSMON_CFG_MON_SEL.RIS and the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_MBWU_CTL access the monitor configuration settings for the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_CFG_MBWU_CTL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance

MPAM	MPAMF_BASE_s	0x0828	MSMON_CFG_MBWU_CTL_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0828	MSMON_CFG_MBWU_CTL_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0828	MSMON_CFG_MBWU_CTL_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0828	MSMON_CFG_MBWU_CTL_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CFG_MBWU_FLT, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register

The MSMON_CFG_MBWU_FLT characteristics are:

Purpose

Controls PARTID and PMG to measure or count in the MBWU monitor selected by MSMON_CFG_MON_SEL.

MSMON_CFG_MBWU_FLT_s sets filter conditions for the Secure memory bandwidth usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_MBWU_CTL_ns sets filter conditions for the Non-secure memory bandwidth usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_FLT_rt sets the filter conditions for the Root PARTID selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_FLT_rl sets the filter conditions for the Realm PARTID selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance filter configuration accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configuration

The power domain of MSMON_CFG_MBWU_FLT is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_CFG_MBWU_FLT are RES0.

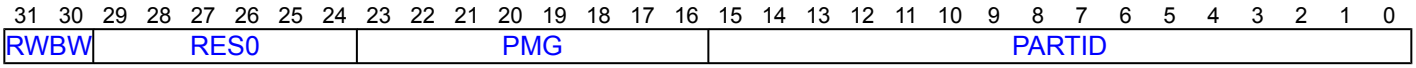
The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_MBWU_FLT is a 32-bit register.

Field descriptions

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:



RW filtering.

RWBW, bits [31:30]

When MPAMF_MBWUMON_IDR.HAS_RWBW == 1:

Read/write bandwidth filter. Configures the selected monitor instance to count all bandwidth, only read bandwidth or only write bandwidth.

RWBW	Meaning
0b00	Monitor instance counts read bandwidth and write bandwidth.
0b01	Monitor instance counts write bandwidth only.
0b10	Monitor instance counts read bandwidth only.
0b11	Reserved.

Otherwise:

Reserved, RES0.

Bits [29:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group to filter memory bandwidth usage monitoring.

If MSMON_CFG_MBWU_CTL.MATCH_PMG == 0, this field is not used to match memory bandwidth to a PMG and the contents of this field is ignored.

If MSMON_CFG_MBWU_CTL.MATCH_PMG == 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts memory bandwidth labeled with PMG equal to this field.

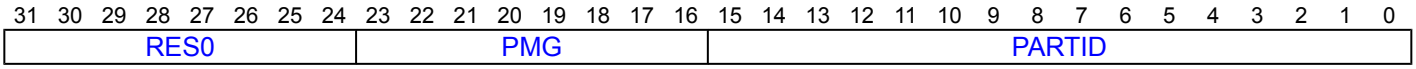
PARTID, bits [15:0]

Partition ID to filter memory bandwidth usage monitoring.

If MSMON_CFG_MBWU_CTL.MATCH_PARTID == 0, this field is not used to match memory bandwidth to a PARTID and the contents of this field is ignored.

If MSMON_CFG_MBWU_CTL.MATCH_PARTID == 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts memory bandwidth labeled with PARTID equal to this field.

Otherwise:



Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group to filter memory bandwidth usage monitoring.

If MSMON_CFG_MBWU_CTL.MATCH_PMG == 0, this field is not used to match memory bandwidth to a PMG and the contents of this field is ignored.

If MSMON_CFG_MBWU_CTL.MATCH_PMG == 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts memory bandwidth labeled with PMG equal to this field.

PARTID, bits [15:0]

Partition ID to filter memory bandwidth usage monitoring.

If MSMON_CFG_MBWU_CTL.MATCH_PARTID == 0, this field is not used to match memory bandwidth to a PARTID and the contents of this field is ignored.

If MSMON_CFG_MBWU_CTL.MATCH_PARTID == 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts memory bandwidth labeled with PARTID equal to this field.

Accessing MSMON_CFG_MBWU_FLT

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CFG_MBWU_FLT_s must only be accessible from the Secure MPAM feature page.
- MSMON_CFG_MBWU_FLT_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_CFG_MBWU_FLT_rt must only be accessible from the Root MPAM feature page.
- MSMON_CFG_MBWU_FLT_rl must only be accessible from the Realm MPAM feature page.

MSMON_CFG_MBWU_FLT_s, MSMON_CFG_MBWU_FLT_ns, MSMON_CFG_MBWU_FLT_rt, and MSMON_CFG_MBWU_FLT_rl must be separate registers:

- The Secure instance (MSMON_CFG_MBWU_FLT_s) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_MBWU_FLT_ns) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_MBWU_FLT_rt) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_CFG_MBWU_FLT_rl) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_MBWU_FLT access the monitor configuration settings for the bandwidth resource instance selected by MSMON_CFG_MON_SEL.RIS and the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_MBWU_FLT access the monitor configuration settings for the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_CFG_MBWU_FLT can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0820	MSMON_CFG_MBWU_FLT_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0820	MSMON_CFG_MBWU_FLT_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0820	MSMON_CFG_MBWU_FLT_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0820	MSMON_CFG_MBWU_FLT_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CFG_MON_SEL, MPAM Monitor Instance Selection Register

The MSMON_CFG_MON_SEL characteristics are:

Purpose

Selects a monitor instance to access through the MSMON configuration and counter registers.

MSMON_CFG_MON_SEL_s selects a Secure monitor instance to access via the Secure MPAM feature page. MSMON_CFG_MON_SEL_ns selects a Non-secure monitor instance to access via the Non-secure MPAM feature page. MSMON_CFG_MON_SEL_rt selects a Root monitor instance to access via the Root MPAM feature page. MSMON_CFG_MON_SEL_rl selects a Realm monitor instance to access via the Realm MPAM feature page.

Note

Different performance monitoring features within an MSC could have different numbers of monitor instances. See the NUM_MON field in the corresponding ID register. This means that a monitor out-of-bounds error might be signaled when an MSMON_CFG register is accessed because the value in MSMON_CFG_MON_SEL.MON_SEL is too large for the particular monitoring feature.

To configure a monitor, set MON_SEL in this register to the index of the monitor instance to configure, then write to the MSMON_CFG_x register to set the configuration of the monitor. At a later time, read the monitor register (for example, MSMON_MBWU) to get the value of the monitor.

Configuration

The power domain of MSMON_CFG_MON_SEL is IMPLEMENTATION DEFINED.

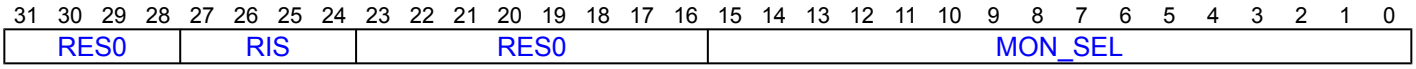
This register is present only when FEAT_MPAM is implemented and (MPAMF_IDR.HAS_MSMON == 1, or (MPAMF_IDR.HAS_IMPL_IDR == 1 and MPAMF_IDR.EXT == 0), or (MPAMF_IDR.HAS_IMPL_IDR == 1, MPAMF_IDR.EXT == 1, and MPAMF_IDR.NO_IMPL_MSMON == 0)). Otherwise, direct accesses to MSMON_CFG_MON_SEL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_MON_SEL is a 32-bit register.

Field descriptions



Bits [31:28]

Reserved, RES0.

RIS, bits [27:24]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented), MPAMF_IDR.EXT == 1, and MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. RIS selects one resource to configure through MSMON_CFG registers.

Otherwise:

Reserved, RES0.

Bits [23:16]

Reserved, RES0.

MON_SEL, bits [15:0]

Selects the monitor instance to configure or read.

Reads and writes to other MSMON registers are indexed by MON_SEL and by the NS bit used to access MSMON_CFG_MON_SEL to access the configuration for a single monitor.

Accessing MSMON_CFG_MON_SEL

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CFG_MON_SEL_s must only be accessible from the Secure MPAM feature page.
- MSMON_CFG_MON_SEL_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_CFG_MON_SEL_rt must only be accessible from the Root MPAM feature page.
- MSMON_CFG_MON_SEL_rl must only be accessible from the Realm MPAM feature page.

MSMON_CFG_MON_SEL_s, MSMON_CFG_MON_SEL_ns, MSMON_CFG_MON_SEL_rt, and MSMON_CFG_MON_SEL_rl must be separate registers:

- The Secure instance (MSMON_CFG_MON_SEL_s) accesses the monitor instance selector used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_MON_SEL_ns) accesses the monitor instance selector used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_MON_SEL_rt) accesses the monitor instance selector used for Root PARTIDs.
- The Realm instance (MSMON_CFG_MON_SEL_rl) accesses the monitor instance selector used for Realm PARTIDs.

MSMON_CFG_MON_SEL can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0800	MSMON_CFG_MON_SEL_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0800	MSMON_CFG_MON_SEL_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0800	MSMON_CFG_MON_SEL_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0800	MSMON_CFG_MON_SEL_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CSU, MPAM Cache Storage Usage Monitor Register

The MSMON_CSU characteristics are:

Purpose

Accesses the CSU monitor instance selected by MSMON_CFG_MON_SEL.

MSMON_CSU_s is a Secure cache storage usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CSU_ns is a Non-secure cache storage usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_CSU_rt is a Root cache storage usage monitor instance selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CSU_rl is a Realm cache storage usage monitor instance selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configuration

The power domain of MSMON_CSU is IMPLEMENTATION DEFINED.

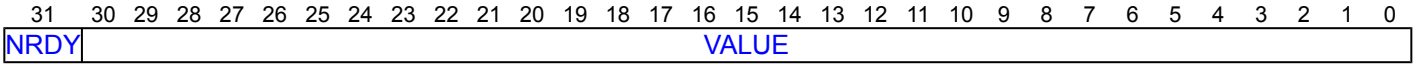
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CSU are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CSU is a 32-bit register.

Field descriptions



NRDY, bit [31]

Not Ready. Indicates whether the monitor instance has possibly inaccurate data.

NRDY	Meaning
0b0	The monitor instance is ready and the MSMON_CSU.VALUE field is accurate.
0b1	The monitor instance is not ready and the contents of the MSMON_CSU.VALUE field might be inaccurate or otherwise not represent the actual cache storage usage.

VALUE, bits [30:0]

Cache storage usage measurement value if MSMON_CSU.NRDY is 0. Invalid if MSMON_CSU.NRDY is 1.

VALUE is the cache storage usage measured in bytes meeting the criteria set in MSMON_CFG_CSU_FLT and MSMON_CFG_CSU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing MSMON_CSU

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CSU_s must only be accessible from the Secure MPAM feature page.
- MSMON_CSU_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_CSU_rt must only be accessible from the Root MPAM feature page.
- MSMON_CSU_rl must only be accessible from the Realm MPAM feature page.

MSMON_CSU_s, MSMON_CSU_ns, MSMON_CSU_rt, and MSMON_CSU_rl must be separate registers:

- The Secure instance (MSMON_CSU_s) accesses the cache storage usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_CSU_ns) accesses the cache storage usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_CSU_rt) accesses the cache storage usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_CSU_rl) accesses the cache storage usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_CSU access the cache storage usage monitor monitor instance for the cache resource instance selected by MSMON_CFG_MON_SEL.RIS and the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_CSU access the cache storage usage monitor monitor instance for the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_CSU can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0840	MSMON_CSU_s

Accessible as follows:

- When MPAMF_CSUMON_IDR.CSU_RO == 0, accesses to this register are **RW**.
- When MPAMF_CSUMON_IDR.CSU_RO == 1, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0840	MSMON_CSU_ns

Accessible as follows:

- When MPAMF_CSUMON_IDR.CSU_RO == 0, accesses to this register are **RW**.
- When MPAMF_CSUMON_IDR.CSU_RO == 1, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0840	MSMON_CSU_rt

Accessible as follows:

- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 0, accesses to this register are **RW**.
- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 1, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0840	MSMON_CSU_rl

Accessible as follows:

- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 0, accesses to this register are **RW**.
- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 1, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CSU_CAPTURE, MPAM Cache Storage Usage Monitor Capture Register

The MSMON_CSU_CAPTURE characteristics are:

Purpose

MSMON_CSU_CAPTURE is a 32-bit read/write register that accesses the captured MSMON_CSU monitor instance selected by MSMON_CFG_MON_SEL.

MSMON_CSU_CAPTURE_s is the Secure cache storage usage monitor capture instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CSU_CAPTURE_ns is the Non-secure cache storage usage monitor capture instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_CSU_CAPTURE_rt is a Root cache storage usage monitor capture instance selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CSU_CAPTURE_rl is a Realm cache storage usage monitor capture instance selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance capture register accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configuration

The power domain of MSMON_CSU_CAPTURE is IMPLEMENTATION DEFINED.

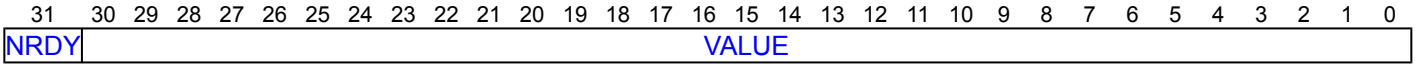
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_CSU == 1, and MPAMF_CSUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_CSU_CAPTURE are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CSU_CAPTURE is a 32-bit register.

Field descriptions



NRDY, bit [31]

Not Ready. Indicates whether the captured monitor value has possibly inaccurate data.

NRDY	Meaning
0b0	The captured monitor instance was ready and the MSMON_CSU_CAPTURE.VALUE field is accurate.
0b1	The captured monitor instance was not ready and the contents of the MSMON_CSU_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual cache storage usage.

VALUE, bits [30:0]

Captured cache storage usage measurement if MSMON_CSU_CAPTURE.NRDY is 0. Invalid if MSMON_CSU_CAPTURE.NRDY is 1.

VALUE is the captured cache storage usage measurement in bytes meeting the criteria set in MSMON_CFG_CSU_FLT and MSMON_CFG_CSU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing MSMON_CSU_CAPTURE

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CSU_CAPTURE_s must only be accessible from the Secure MPAM feature page.
- MSMON_CSU_CAPTURE_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_CSU_CAPTURE_rt must only be accessible from the Root MPAM feature page.
- MSMON_CSU_CAPTURE_rl must only be accessible from the Realm MPAM feature page.

MSMON_CSU_CAPTURE_s, MSMON_CSU_CAPTURE_ns, MSMON_CSU_CAPTURE_rt, and MSMON_CSU_CAPTURE_rl must be separate registers:

- The Secure instance (MSMON_CSU_CAPTURE_s) accesses the captured cache storage usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_CSU_CAPTURE_ns) accesses the captured cache storage usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_CSU_CAPTURE_rt) accesses the captured cache storage usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_CSU_CAPTURE_rl) accesses the captured cache storage usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_CSU_CAPTURE access the monitor instance for the cache resource instance selected by MSMON_CFG_MON_SEL.RIS and the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_CSU_CAPTURE access the monitor instance for the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_CSU_CAPTURE can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0848	MSMON_CSU_CAPTURE_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0848	MSMON_CSU_CAPTURE_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0848	MSMON_CSU_CAPTURE_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0848	MSMON_CSU_CAPTURE_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CSU_OFSR, MPAM CSU Monitor Overflow Status Register

The MSMON_CSU_OFSR characteristics are:

Purpose

MSMON_CSU_OFSR is a 32-bit read-only register that shows bitmap of CSU monitor instance overflow status for a contiguous group of 32 monitor instances.

MSMON_CSU_OFSR_s gives a bitmap of pending CSU overflow status for 32 Secure CSU monitor instances. MSMON_CSU_OFSR_ns gives a bitmap of pending CSU overflow status for 32 Non-secure CSU monitor instances. MSMON_CSU_OFSR_rt gives a bitmap of pending CSU overflow status for 32 Root CSU monitor instances. MSMON_CSU_OFSR_rl gives a bitmap of pending CSU overflow status for 32 Realm CSU monitor instances.

Configuration

The power domain of MSMON_CSU_OFSR is IMPLEMENTATION DEFINED.

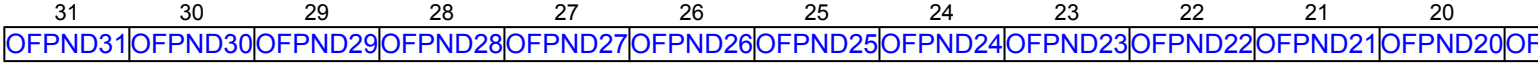
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_CSU == 1, and MPAMF_CSUMON_IDR.HAS_OFSR == 1. Otherwise, direct accesses to MSMON_CSU_OFSR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CSU_OFSR is a 32-bit register.

Field descriptions



OFPND<i>, bit [i], for i = 31 to 0

Overflow status bitmap for CSU monitor instances. The RIS and the contiguous range of CSU monitor instances are set in MSMON_CFG_MON_SEL. i of 0 corresponds to the CSU monitor instance MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0.

OFPND<i>	Meaning
0b0	CSU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0 + i) does not have a pending overflow.
0b1	CSU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0 + i) has a pending overflow.

After reading MSMON_OFLOW_SR to determine that a CSU monitor instance has a pending overflow and which RIS values have pending overflows, an interrupt service routine could poll groups of 32 monitor instances in a RIS for pending monitors by reading this bitmap and incrementing MSMON_CFG_MON_SEL.MON_SEL by 32.

Accessing MSMON_CSU_OFSR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CSU_OFSR_s must only be accessible from the Secure MPAM feature page.
- MSMON_CSU_OFSR_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_CSU_OFSR_rt must only be accessible from the Root MPAM feature page.

- MSMON_CSU_OFSR_rl must only be accessible from the Realm MPAM feature page.

MSMON_CSU_OFSR_s, MSMON_CSU_OFSR_ns, MSMON_CSU_OFSR_rt, and MSMON_CSU_OFSR_rl must be separate registers:

- The Secure instance (MSMON_CSU_OFSR_s) accesses the CSU monitor overflow status bitmap used for Secure PARTIDs.
- The Non-secure instance (MSMON_CSU_OFSR_ns) accesses the CSU monitor overflow status bitmap used for Non-secure PARTIDs.
- The Root instance (MSMON_CSU_OFSR_rt) accesses the CSU monitor overflow status bitmap used for Root PARTIDs.
- The Realm instance (MSMON_CSU_OFSR_rl) accesses the CSU monitor overflow status bitmap used for Realm PARTIDs.

MSMON_CSU_OFSR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0858	MSMON_CSU_OFSR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0858	MSMON_CSU_OFSR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0858	MSMON_CSU_OFSR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0858	MSMON_CSU_OFSR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_MBWU, MPAM Memory Bandwidth Usage Monitor Register

The MSMON_MBWU characteristics are:

Purpose

Accesses the monitor instance selected by MSMON_CFG_MON_SEL.

MSMON_MBWU_s is the Secure memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_s. MSMON_MBWU_ns is the Non-secure memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_ns. MSMON_MBWU_rt is the Root memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_rt. MSMON_MBWU_rl is the Realm memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_rl.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance register accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configuration

The power domain of MSMON_MBWU is IMPLEMENTATION DEFINED.

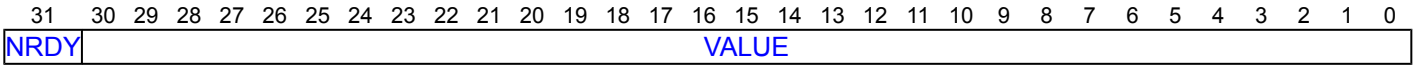
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_MBWU are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU is a 32-bit register.

Field descriptions



NRDY, bit [31]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

NRDY	Meaning
0b0	The monitor instance is ready and the MSMON_MBWU.VALUE field is accurate.
0b1	The monitor instance is not ready and the contents of the MSMON_MBWU.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

VALUE, bits [30:0]

Memory bandwidth usage counter value if MSMON_MBWU.NRDY is 0. Invalid if MSMON_MBWU.NRDY is 1.

VALUE is the scaled count of bytes transferred since the monitor was last reset that met the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

If MSMON_CFG_MBWU_CTL.SCLEN enables scaling, the count in VALUE is the number of bytes shifted right by MPAMF_MBWUMON_IDR.SCALE bit positions and rounded.

Accessing MSMON_MBWU

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_MBWU_s must only be accessible from the Secure MPAM feature page.
- MSMON_MBWU_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_rt must only be accessible from the Root MPAM feature page.
- MSMON_MBWU_rl must only be accessible from the Realm MPAM feature page.

MSMON_MBWU_s, MSMON_MBWU_ns, MSMON_MBWU_rt, and MSMON_MBWU_rl must be separate registers:

- The Secure instance (MSMON_MBWU_s) accesses the memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_ns) accesses the memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_rt) accesses the memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_rl) accesses the memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU access the memory bandwidth usage monitor instance for the resource instance selected by MSMON_CFG_MON_SEL.RIS and the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU access the memory bandwidth usage monitor instance for the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_MBWU can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0860	MSMON_MBWU_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0860	MSMON_MBWU_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0860	MSMON_MBWU_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0860	MSMON_MBWU_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_MBWU_CAPTURE, MPAM Memory Bandwidth Usage Monitor Capture Register

The MSMON_MBWU_CAPTURE characteristics are:

Purpose

Accesses the captured MSMON_MBWU monitor instance selected by MSMON_CFG_MON_SEL.

MSMON_MBWU_CAPTURE_s is the Secure memory bandwidth usage monitor capture instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_CAPTURE_ns is the Non-secure memory bandwidth usage monitor capture instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_CAPTURE_rt is the Root memory bandwidth usage monitor capture instance selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_MBWU_CAPTURE_rl is the Realm memory bandwidth usage monitor capture instance selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance capture register accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configuration

The power domain of MSMON_MBWU_CAPTURE is IMPLEMENTATION DEFINED.

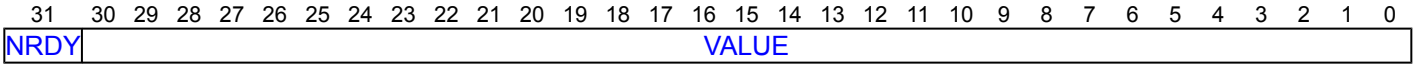
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1, and MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_MBWU_CAPTURE are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_CAPTURE is a 32-bit register.

Field descriptions



NRDY, bit [31]

Not Ready. The captured NRDY bit from the corresponding instance of MSMON_MBWU. This bit indicates whether the captured monitor value has possibly inaccurate data.

NRDY	Meaning
0b0	The captured monitor instance was ready and the MSMON_MBWU_CAPTURE.VALUE field is accurate.
0b1	The captured monitor instance was not ready and the contents of the MSMON_MBWU_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

VALUE, bits [30:0]

Captured memory bandwidth usage counter value if MSMON_MBWU_CAPTURE.NRDY is 0. Invalid if MSMON_MBWU_CAPTURE.NRDY is 1.

VALUE is the captured VALUE field from the corresponding instance of MSMON_MBWU, the count of bytes transferred since the monitor was last reset that meet the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

VALUE captures the MSMON_MBWU.VALUE and preserves any scaling that had been performed on the VALUE field in that register.

Accessing MSMON_MBWU_CAPTURE

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_MBWU_CAPTURE_s must only be accessible from the Secure MPAM feature page.
- MSMON_MBWU_CAPTURE_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_CAPTURE_rt must only be accessible from the Root MPAM feature page.
- MSMON_MBWU_CAPTURE_rl must only be accessible from the Realm MPAM feature page.

MSMON_MBWU_CAPTURE_s, MSMON_MBWU_CAPTURE_ns, MSMON_MBWU_CAPTURE_rt, and MSMON_MBWU_CAPTURE_rl must be separate registers:

- The Secure instance (MSMON_MBWU_CAPTURE_s) accesses the captured memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_CAPTURE_ns) accesses the captured memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_CAPTURE_rt) accesses the captured memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_CAPTURE_rl) accesses the captured memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU_CAPTURE access the monitor instance for the bandwidth resource instance selected by MSMON_CFG_MON_SEL.RIS and the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU_CAPTURE access the monitor instance for the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_MBWU_CAPTURE can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0868	MSMON_MBWU_CAPTURE_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0868	MSMON_MBWU_CAPTURE_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0868	MSMON_MBWU_CAPTURE_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0868	MSMON_MBWU_CAPTURE_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_MBWU_L, MPAM Long Memory Bandwidth Usage Monitor Register

The MSMON_MBWU_L characteristics are:

Purpose

Accesses the monitor instance selected by MSMON_CFG_MON_SEL.

MSMON_MBWU_L_s is the Secure long memory bandwidth usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_L_ns is the Non-secure long memory bandwidth usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_L_rt is the Root long memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_rt. MSMON_MBWU_L_rl is the Realm long memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_rl.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance long monitor register accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configuration

The power domain of MSMON_MBWU_L is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1, and MPAMF_MBWUMON_IDR.HAS_LONG == 1. Otherwise, direct accesses to MSMON_MBWU_L are RES0.

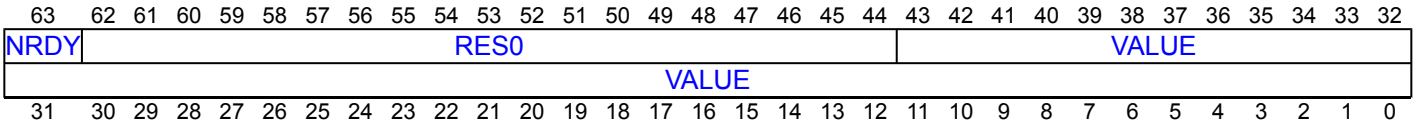
The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_L is a 64-bit register.

Field descriptions

When MPAMF_MBWUMON_IDR.LWD == 0:



NRDY, bit [63]

Not Ready. Indicates whether the monitor instance has possibly inaccurate data.

NRDY	Meaning
0b0	The monitor instance is ready and the MSMON_MBWU_L.VALUE field is accurate.
0b1	The monitor instance is not ready and the contents of the MSMON_MBWU_L.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

Bits [62:44]

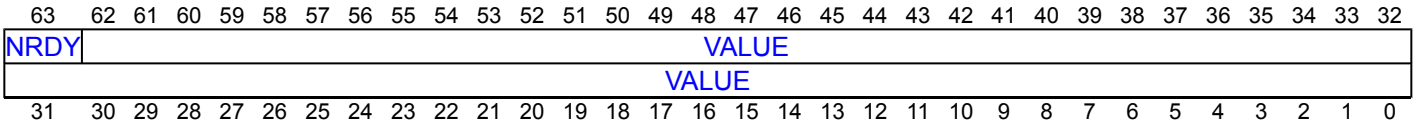
Reserved, RES0.

VALUE, bits [43:0]

Long (44-bit) memory bandwidth usage counter value if MSMON_MBWU_L.NRDY is 0. Invalid if MSMON_MBWU_L.NRDY is 1.

VALUE is the long count of bytes transferred since the monitor was last reset that met the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

When MPAMF_MBWUMON_IDR.LWD == 1:



NRDY, bit [63]

Not Ready. Indicates whether the monitor instance has possibly inaccurate data.

NRDY	Meaning
0b0	The monitor instance is ready and the MSMON_MBWU_L.VALUE field is accurate.
0b1	The monitor instance is not ready and the contents of the MSMON_MBWU_L.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

VALUE, bits [62:0]

Long (63-bit) memory bandwidth usage counter value if MSMON_MBWU_L.NRDY is 0. Invalid if MSMON_MBWU_L.NRDY is 1.

VALUE is the long count of bytes transferred since the monitor instance was last reset that met the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing MSMON_MBWU_L

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_MBWU_L_s must only be accessible from the Secure MPAM feature page.
- MSMON_MBWU_L_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_L_rt must only be accessible from the Root MPAM feature page.
- MSMON_MBWU_L_rl must only be accessible from the Realm MPAM feature page.

MSMON_MBWU_L_s, MSMON_MBWU_L_ns, MSMON_MBWU_L_rt, and MSMON_MBWU_L_rl must be separate registers:

- The Secure instance (MSMON_MBWU_L_s) accesses the long memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_L_ns) accesses the long memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_L_rt) accesses the long memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_L_rl) accesses the long memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU_L access the long memory bandwidth usage monitor instance for the bandwidth resource instance selected by MSMON_CFG_MON_SEL.RIS and the monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU_L access the long memory bandwidth usage monitor instance for the monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_MBWU_L can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0880	MSMON_MBWU_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0880	MSMON_MBWU_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0880	MSMON_MBWU_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0880	MSMON_MBWU_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_MBWU_L_CAPTURE, MPAM Long Memory Bandwidth Usage Monitor Capture Register

The MSMON_MBWU_L_CAPTURE characteristics are:

Purpose

Accesses the captured MSMON_MBWU_L monitor instance selected by MSMON_CFG_MON_SEL.

MSMON_MBWU_L_CAPTURE_s is the Secure long memory bandwidth usage monitor capture instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_L_CAPTURE_ns is the Non-secure long memory bandwidth usage monitor capture instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_L_CAPTURE_rt is the Root long memory bandwidth usage monitor capture instance selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_MBWU_L_CAPTURE_rl is the Realm long memory bandwidth usage monitor capture instance selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance long capture register accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configuration

The power domain of MSMON_MBWU_L_CAPTURE is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1, MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1, and MPAMF_MBWUMON_IDR.HAS_LONG == 1. Otherwise, direct accesses to MSMON_MBWU_L_CAPTURE are RES0.

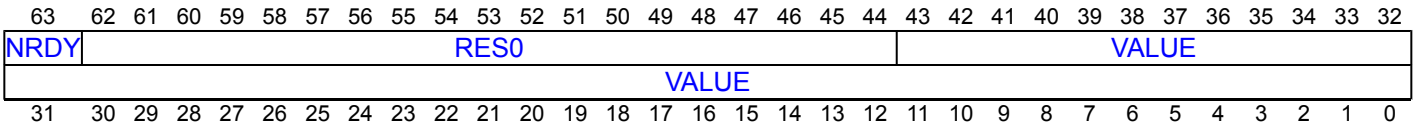
The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_L_CAPTURE is a 64-bit register.

Field descriptions

When MPAMF_MBWUMON_IDR.LWD == 0:



NRDY, bit [63]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

NRDY	Meaning
0b0	The captured monitor instance was ready and the MSMON_MBWU_L_CAPTURE.VALUE field is accurate.
0b1	The captured monitor instance was not ready and the contents of the MSMON_MBWU_L_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

Bits [62:44]

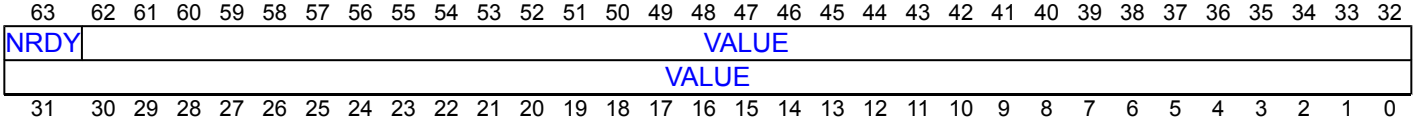
Reserved, RES0.

VALUE, bits [43:0]

Captured long memory bandwidth usage counter value if MSMON_MBWU_L_CAPTURE.NRDY is 0. Invalid if MSMON_MBWU_L_CAPTURE.NRDY is 1.

VALUE is the captured 44-bit count of bytes transferred since the monitor instance was last reset that met the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

When MPAMF_MBWUMON_IDR.LWD == 1:



NRDY, bit [63]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

NRDY	Meaning
0b0	The captured monitor instance was ready and the MSMON_MBWU_L_CAPTURE.VALUE field is accurate.
0b1	The captured monitor instance was not ready and the contents of the MSMON_MBWU_L_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

VALUE, bits [62:0]

The captured long memory bandwidth usage counter value if MSMON_MBWU_L_CAPTURE.NRDY is 0. Invalid if MSMON_MBWU_L_CAPTURE.NRDY is 1.

VALUE is the captured 63-bit count of bytes transferred since the monitor instance was last reset that met the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing MSMON_MBWU_L_CAPTURE

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_MBWU_L_CAPTURE_s must only be accessible from the Secure MPAM feature page.
- MSMON_MBWU_L_CAPTURE_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_L_CAPTURE_rt must only be accessible from the Root MPAM feature page.
- MSMON_MBWU_L_CAPTURE_rl must only be accessible from the Realm MPAM feature page.

MSMON_MBWU_L_CAPTURE_s, MSMON_MBWU_L_CAPTURE_ns, MSMON_MBWU_L_CAPTURE_rt, and MSMON_MBWU_L_CAPTURE_rl must be separate registers:

- The Secure instance (MSMON_MBWU_L_CAPTURE_s) accesses the captured long memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_L_CAPTURE_ns) accesses the captured long memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_L_CAPTURE_rt) accesses the captured long memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_L_CAPTURE_rl) accesses the captured long memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU_L_CAPTURE access the monitor instance for the bandwidth resource instance selected by MSMON_CFG_MON_SEL.RIS and the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU_L_CAPTURE access the monitor instance for the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_MBWU_L_CAPTURE can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0890	MSMON_MBWU_CAPTURE_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0890	MSMON_MBWU_CAPTURE_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0890	MSMON_MBWU_CAPTURE_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0890	MSMON_MBWU_CAPTURE_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_MBWU_OFSR, MPAM MBWU Monitor Overflow Status Register

The MSMON_MBWU_OFSR characteristics are:

Purpose

MSMON_MBWU_OFSR is a 32-bit read-only register that shows bitmap of MBWU monitor instance overflow status for a contiguous group of 32 monitor instances.

MSMON_MBWU_OFSR_s gives a bitmap of pending MBWU overflow status for 32 Secure MBWU monitor instances.
MSMON_MBWU_OFSR_ns gives a bitmap of pending MBWU overflow status for 32 Non-secure MBWU monitor instances.
MSMON_MBWU_OFSR_rt gives a bitmap of pending MBWU overflow status for 32 Root MBWU monitor instances. MSMON_MBWU_OFSR_rl gives a bitmap of pending MBWU overflow status for 32 Realm MBWU monitor instances.

Configuration

The power domain of MSMON_MBWU_OFSR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1, and MPAMF_MBWUMON_IDR.HAS_OFSR == 1. Otherwise, direct accesses to MSMON_MBWU_OFSR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_OFSR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	
OFPND31	OFPND30	OFPND29	OFPND28	OFPND27	OFPND26	OFPND25	OFPND24	OFPND23	OFPND22	OFPND21	OFPND20	OF

OFPND<i>, bit [i], for i = 31 to 0

Overflow status bitmap for MBWU monitor instances. The RIS and the contiguous range of MBWU monitor instances are set in MSMON_CFG_MON_SEL. i of 0 corresponds to the MBWU monitor instance MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0.

OFPND<i>	Meaning
0b0	MBWU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0 + i) does not have a pending overflow.
0b1	MBWU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0 + i) has a pending overflow.

After reading MSMON_OFLOW_SR to determine that an MBWU monitor instance has a pending overflow and which RIS values have pending overflows, an interrupt service routine could poll groups of 32 monitor instances in a RIS for pending monitors by reading this bitmap and incrementing MSMON_CFG_MON_SEL.MON_SEL by 32.

A pending overflow may be in either the MSMON_CFG_MBWU_CTL.OFLOW_STATUS or MSMON_CFG_MBWU_CTL.OFLOW_STATUS_L field.

Accessing MSMON_MBWU_OFSR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_MBWU_OFSR_s must only be accessible from the Secure MPAM feature page.
- MSMON_MBWU_OFSR_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_OFSR_rt must only be accessible from the Root MPAM feature page.
- MSMON_MBWU_OFSR_rl must only be accessible from the Realm MPAM feature page.

MSMON_MBWU_OFSR_s, MSMON_MBWU_OFSR_ns, MSMON_MBWU_OFSR_rt, and MSMON_MBWU_OFSR_rl must be separate registers:

- The Secure instance (MSMON_MBWU_OFSR_s) accesses the MBWU monitor overflow status bitmap used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_OFSR_ns) accesses the MBWU monitor overflow status bitmap used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_OFSR_rt) accesses the MBWU monitor overflow status bitmap used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_OFSR_rl) accesses the MBWU monitor overflow status bitmap used for Realm PARTIDs.

MSMON_MBWU_OFSR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0898	MSMON_MBWU_OFSR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0898	MSMON_MBWU_OFSR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0898	MSMON_MBWU_OFSR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0898	MSMON_MBWU_OFSR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_OFLOW_MSI_ADDR_H, MPAM Monitor Overflow MSI Write High-part Address Register

The MSMON_OFLOW_MSI_ADDR_H characteristics are:

Purpose

MSMON_OFLOW_MSI_ADDR_H is a 32-bit read/write register for the high part of the MPAM monitor overflow MSI address.

MSMON_OFLOW_MSI_ADDR_H_s is the high part of the MSI write address for monitor overflow interrupts from Secure monitor instances.
MSMON_OFLOW_MSI_ADDR_H_ns is the high part of the MSI write address for monitor overflow interrupts from Non-secure monitor instances.
MSMON_OFLOW_MSI_ADDR_H_rt is the high part of the MSI write address for monitor overflow interrupts from Root monitor instances.
MSMON_OFLOW_MSI_ADDR_H_rl is the high part of the MSI write address for monitor overflow interrupts from Realm monitor instances.

Configuration

The power domain of MSMON_OFLOW_MSI_ADDR_H is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_ADDR_H are RES0.

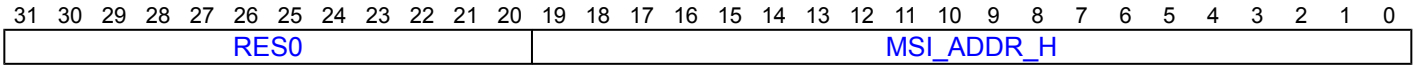
MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA, and MSMON_OFLOW_MSI_MPAM must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_ADDR_H is a 32-bit register.

Field descriptions



Bits [31:20]

Reserved, RES0.

MSI_ADDR_H, bits [19:0]

MSI write address bits[51:32].

Accessing MSMON_OFLOW_MSI_ADDR_H

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLW_MSI_ADDR_H_s must only be accessible from the Secure MPAM feature page.
- MSMON_OFLW_MSI_ADDR_H_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_OFLW_MSI_ADDR_H_rt must only be accessible from the Root MPAM feature page.
- MSMON_OFLW_MSI_ADDR_H_rl must only be accessible from the Realm MPAM feature page.

MSMON_OFLW_MSI_ADDR_H_s, MSMON_OFLW_MSI_ADDR_H_ns, MSMON_OFLW_MSI_ADDR_H_rt, and MSMON_OFLW_MSI_ADDR_H_rl must be separate registers:

- The Secure instance (MSMON_OFLW_MSI_ADDR_H_s) accesses the high part of the monitor overflow MSI write address of Secure monitors.
- The Non-secure instance (MSMON_OFLW_MSI_ADDR_H_ns) accesses the high part of the monitor overflow MSI write address of Non-secure monitors.
- The Root instance (MSMON_OFLW_MSI_ADDR_H_rt) accesses the high part of the monitor overflow MSI write address of Root monitors.
- The Realm instance (MSMON_OFLW_MSI_ADDR_H_rl) accesses the high part of the monitor overflow MSI write address of Realm monitors.

MSMON_OFLOW_MSI_ADDR_H can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x08E4	MSMON_OFLW_MSI_ADDR_H_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x08E4	MSMON_OFLW_MSI_ADDR_H_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x08E4	MSMON_OFLW_MSI_ADDR_H_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08E4	MSMON_OFLW_MSI_ADDR_H_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_OFLOW_MSI_ADDR_L, MPAM Monitor Overflow MSI Low-part Address Register

The MSMON_OFLOW_MSI_ADDR_L characteristics are:

Purpose

MSMON_OFLOW_MSI_ADDR_L is a 32-bit read/write register for the low part of the MPAM monitor MSI address.

MSMON_OFLOW_MSI_ADDR_L_s is the low part of the MSI write address for overflow interrupts from Secure monitor intances.
MSMON_OFLOW_MSI_ADDR_L_ns is the low part of the MSI write address for overflow interrupts from Non-secure monitor instances.
MSMON_OFLOW_MSI_ADDR_L_rt is the low part of the MSI write address for overflow interrupts from Root monitor intances.
MSMON_OFLOW_MSI_ADDR_L_rl is the low part of the MSI write address for overflow interrupts from Realm monitor instances.

Configuration

The power domain of MSMON_OFLOW_MSI_ADDR_L is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_ADDR_L are RES0.

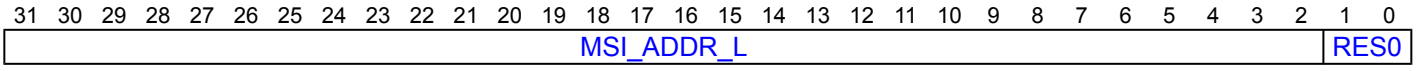
MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA, and MSMON_OFLOW_MSI_MPAM must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_ADDR_L is a 32-bit register.

Field descriptions



MSI_ADDR_L, bits [31:2]

MSI write address bits[31:2].

Bits [1:0]

Reserved, RES0.

Accessing MSMON_OFLOW_MSI_ADDR_L

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLOW_MSI_ADDR_L_s must only be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_L_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_L_rt must only be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_L_rl must only be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_ADDR_L_s, MSMON_OFLOW_MSI_ADDR_L_ns, MSMON_OFLOW_MSI_ADDR_L_rt, and MSMON_OFLOW_MSI_ADDR_L_rl must be separate registers:

- The Secure instance (MSMON_OFLOW_MSI_ADDR_L_s) accesses the low part of the overflow MSI write address used for Secure PARTIDs.
- The Non-secure instance (MSMON_OFLOW_MSI_ADDR_L_ns) accesses the low part of the overflow MSI write address used for Non-secure PARTIDs.
- The Root instance (MSMON_OFLOW_MSI_ADDR_L_rt) accesses the low part of the overflow MSI write address used for Root PARTIDs.
- The Realm instance (MSMON_OFLOW_MSI_ADDR_L_rl) accesses the low part of the overflow MSI write address used for Realm PARTIDs.

MSMON_OFLOW_MSI_ADDR_L can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x08E0	MSMON_OFLOW_MSI_ADDR_L_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x08E0	MSMON_OFLOW_MSI_ADDR_L_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x08E0	MSMON_OFLOW_MSI_ADDR_L_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08E0	MSMON_OFLOW_MSI_ADDR_L_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_OFLOW_MSI_ATTR, MPAM Monitor Overflow MSI Write Attributes Register

The MSMON_OFLOW_MSI_ATTR characteristics are:

Purpose

MSMON_OFLOW_MSI_ATTR is a 32-bit read/write register that controls MPAM monitor overflow MSI write attributes for MPAM monitor overflows in this MSC.

MSMON_OFLOW_MSI_ATTR_s controls Secure MPAM monitor overflow MSI writes. MSMON_OFLOW_MSI_ATTR_ns controls Non-secure MPAM monitor overflow MSI writes. MSMON_OFLOW_MSI_ATTR_rt controls Root MPAM monitor overflow MSI writes. MSMON_OFLOW_MSI_ATTR_rl controls Realm MPAM monitor overflow MSI writes.

Configuration

The power domain of MSMON_OFLOW_MSI_ATTR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_ATTR are RES0.

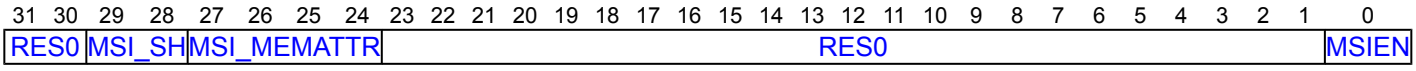
MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA, and MSMON_OFLOW_MSI_MPAM must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_ATTR is a 32-bit register.

Field descriptions



Bits [31:30]

Reserved, RES0.

MSI_SH, bits [29:28]

Sharability attribute of MSI writes.

MSI_SH	Meaning
0b00	Non-shareable.
0b01	Reserved, CONSTRAINED UNPREDICTABLE.
0b10	Outer Shareable.
0b11	Inner Shareable.

When MSMON_OFLOW_MSI_ATTR.MSI_MEMATTR specifies a Device memory type, the contents of this field are IGNORED and Shareability is effectively Outer Shareable.

MSI_MEMATTR, bits [27:24]

Memory attributes of MSI writes.

Note

This encoding matches the VMSAv8-64 stage 2 MemAttr[3:0] field as described in the Arm ARM, except that the following encodings are Reserved (not UNPREDICTABLE) and behave as DEvice-nGnRnE: `0b0100`, `0b1000`, and `0b1100`.

MSI_MEMATTR	Meaning
`0b0000`	Device-nGnRnE.
`0b0001`	Device-nGnRE.
`0b0010`	Device-nGRE.
`0b0011`	Device-GRE.
`0b0100`	Reserved. Behave as Device-nGnRnE, `0b0000`.
`0b0101`	Normal Inner Non-cacheable, Outer Non-cacheable.
`0b0110`	Normal Inner Write-Through Cacheable, Outer Non-cacheable.
`0b0111`	Normal Inner Write-Back Cacheable, Outer Non-cacheable.
`0b1000`	Reserved. Behave as Device-nGnRnE, `0b0000`.
`0b1001`	Normal Inner Non-Cachable, Outer Write-Through Cacheable.
`0b1010`	Normal Inner Write-Through Cacheable, Outer Write-Through Cachable.
`0b1011`	Normal Inner Write-Back Cacheable, Outer Write-Through Cachable.
`0b1100`	Reserved. Behave as Device-nGnRnE, `0b0000`.
`0b1101`	Normal Inner Non-cacheable, Outer Write-Back Cacheable.
`0b1110`	Normal Inner Write-Through Cacheable, Outer Write-Back Cacheable.
`0b1111`	Normal Inner Write-Back Cacheable, Outer Write-Back Cacheable.

When this field specifies a Device memory type, the contents of MSMON_OFLOW_MSI_ATTR.MSI_SH are IGNORED and Shareability is effectively Outer Shareable.

Device types may be implemented as any Device type with more n characters. For example, if this field is set to `0b0010`, an implementation may treat the MSI write as the specified type, Device-nGRE, or as Device-nGnRE or as Device-nGnRnE.

Reserved encodings `0b0100`, `0b1000`, and `0b1100` must be implemented to behave the same as the `0b0000` encoding.

Bits [23:1]

Reserved, RES0.

MSIEN, bit [0]

Monitor overflow MSI write enable.

MSIEN	Meaning
`0b0`	MPAM monitor overflow MSI writes are not generated to signal enabled MPAM monitor overflow interrupts. When monitor overflow MSI writes are disabled, hardwired monitor overflow interrupt could be generated if hardwired monitor overflow interrupt is implemented.
`0b1`	MPAM monitor overflow MSI writes are generated to signal enabled MPAM monitor overflow interrupts. When monitor overflow MSI writes are enabled, hardwired monitor overflow interrupts are not generated.

This enable affects whether a hardwired overlow interrupt is generated.

The reset behavior of this field is:

- On a MSC reset, this field resets to `'0'`.

Accessing MSMON_OFLOW_MSI_ATTR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLOW_MSI_ATTR_s must only be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_ATTR_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_ATTR_rt must only be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_ATTR_rl must only be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_ATTR_s, MSMON_OFLOW_MSI_ATTR_ns, MSMON_OFLOW_MSI_ATTR_rt, and MSMON_OFLOW_MSI_ATTR_rl must be separate registers:

- The Secure instance (MSMON_OFLOW_MSI_ATTR_s) accesses the monitor overflow MSI write attributes of Secure monitors.
- The Non-secure instance (MSMON_OFLOW_MSI_ATTR_ns) accesses the monitor overflow MSI write attributes of Non-secure monitors.
- The Root instance (MSMON_OFLOW_MSI_ATTR_rt) accesses the monitor overflow MSI write attributes of Root monitors.
- The Realm instance (MSMON_OFLOW_MSI_ATTR_rl) accesses the monitor overflow MSI write attributes of Realm monitors.

MSMON_OFLOW_MSI_ATTR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x08EC	MSMON_OFLOW_MSI_ATTR_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x08EC	MSMON_OFLOW_MSI_ATTR_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x08EC	MSMON_OFLOW_MSI_ATTR_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08EC	MSMON_OFLOW_MSI_ATTR_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_OFLOW_MSI_DATA, MPAM Monitor Overflow MSI Write Data Register

The MSMON_OFLOW_MSI_DATA characteristics are:

Purpose

MSMON_OFLOW_MSI_DATA is a 32-bit read/write register for the MPAM monitor overflow MSI data.

MSMON_OFLOW_MSI_DATA_s is the data for the MSI write for monitor overflow from Secure monitor instances.
MSMON_OFLOW_MSI_DATA_ns is the data for the MSI writes for monitor overflow interrupts from Non-secure monitor instances.
MSMON_OFLOW_MSI_DATA_rt is the data for the MSI write for monitor overflow from Root monitor instances.
MSMON_OFLOW_MSI_DATA_rl is the data for the MSI writes for monitor overflow interrupts from Realm monitor instances.

Configuration

The power domain of MSMON_OFLOW_MSI_DATA is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_DATA are RES0.

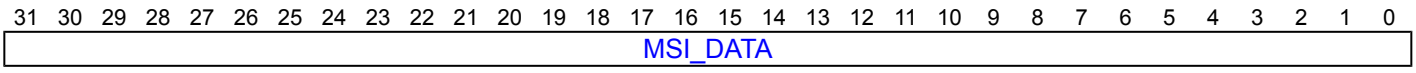
MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA, and MSMON_OFLOW_MSI_MPAM must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_DATA is a 32-bit register.

Field descriptions



MSI_DATA, bits [31:0]

MSI write data word.

Accessing MSMON_OFLOW_MSI_DATA

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLOW_MSI_DATA_s must only be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_DATA_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_DATA_rt must only be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_DATA_rl must only be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_DATA_s, MSMON_OFLOW_MSI_DATA_ns, MSMON_OFLOW_MSI_DATA_rt, and MSMON_OFLOW_MSI_DATA_rl must be separate registers:

- The Secure instance (MSMON_OFLOW_MSI_DATA_s) accesses the monitor overflow MSI write data of Secure monitors.
- The Non-secure instance (MSMON_OFLOW_MSI_DATA_ns) accesses the monitor overflow MSI write data of Non-secure monitors.
- The Root instance (MSMON_OFLOW_MSI_DATA_rt) accesses the monitor overflow MSI write data of Root monitors.
- The Realm instance (MSMON_OFLOW_MSI_DATA_rl) accesses the monitor overflow MSI write data of Realm monitors.

MSMON_OFLOW_MSI_DATA can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x08E8	MSMON_OFLOW_MSI_DATA_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x08E8	MSMON_OFLOW_MSI_DATA_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x08E8	MSMON_OFLOW_MSI_DATA_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08E8	MSMON_OFLOW_MSI_DATA_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_OFLOW_MSI_MPAM, MPAM Monitor Overflow MSI Write MPAM Information Register

The MSMON_OFLOW_MSI_MPAM characteristics are:

Purpose

MSMON_OFLOW_MSI_MPAM is a 32-bit read/write register that sets the MPAM information for a monitor overflow MSI write.

MSMON_OFLOW_MSI_MPAM_s controls MPAM information labeling of Secure monitor overflow MSI writes.
MSMON_OFLOW_MSI_MPAM_ns controls MPAM information labeling of Non-secure monitor overflow MSI writes.
MSMON_OFLOW_MSI_MPAM_rt controls MPAM information labeling of Root monitor overflow MSI writes. MSMON_OFLOW_MSI_MPAM_rl controls MPAM information labeling of Realm monitor overflow MSI writes.

Configuration

The power domain of MSMON_OFLOW_MSI_MPAM is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_MPAM are RES0.

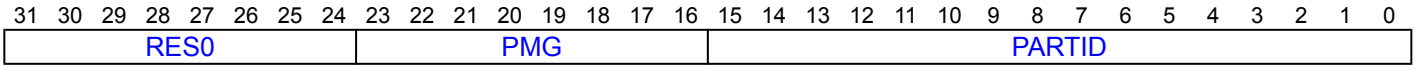
MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA, and MSMON_OFLOW_MSI_MPAM must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_MPAM is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group property for an MSC monitor overflow MSI write.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

PARTID, bits [15:0]

Partition ID for an MSC monitor overflow MSI write.

The PARTID in this field is in the Secure PARTID space in the MSMON_OFLOW_MSI_MPAM_s instance and in the Non-secure PARTID space in the MSMON_OFLOW_MSI_MPAM_ns instance of this register.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

Accessing MSMON_OFLOW_MSI_MPAM

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLOW_MSI_MPAM_s must only be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_MPAM_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_MPAM_rt must only be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_MPAM_rl must only be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_MPAM_s, MSMON_OFLOW_MSI_MPAM_ns, MSMON_OFLOW_MSI_MPAM_rt, and MSMON_OFLOW_MSI_MPAM_rl must be separate registers:

- The Secure instance (MSMON_OFLOW_MSI_MPAM_s) accesses the monitor overflow MSI MPAM information of Secure monitors.
- The Non-secure instance (MSMON_OFLOW_MSI_MPAM_ns) accesses the monitor overflow MSI MPAM information of Non-secure monitors.
- The Root instance (MSMON_OFLOW_MSI_MPAM_rt) accesses the monitor overflow MSI MPAM information of Root monitors.
- The Realm instance (MSMON_OFLOW_MSI_MPAM_rl) accesses the monitor overflow MSI MPAM information of Realm monitors.

MSMON_OFLOW_MSI_MPAM can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x08DC	MSMON_OFLOW_MSI_MPAM_s

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x08DC	MSMON_OFLOW_MSI_MPAM_ns

Accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x08DC	MSMON_OFLOW_MSI_MPAM_rt

When FEAT_RME is implemented, accesses to this register are **RW**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08DC	MSMON_OFLOW_MSI_MPAM_rl

When FEAT_RME is implemented, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_OFLOW_SR, MPAM Monitor Overflow Status Register

The MSMON_OFLOW_SR characteristics are:

Purpose

MSMON_OFLOW_SR is a 32-bit read-only register that shows MPAM monitor overflow status for this MSC.

MSMON_OFLOW_SR_s gives the status of overflows of Secure MPAM monitors. MSMON_OFLOW_SR_ns gives the status of overflows of Non-secure MPAM monitors. MSMON_OFLOW_SR_rt gives the status of overflows of Root MPAM monitors. MSMON_OFLOW_SR_rl gives the status of overflows of Realm MPAM monitors.

Configuration

The power domain of MSMON_OFLOW_SR is IMPLEMENTATION DEFINED.

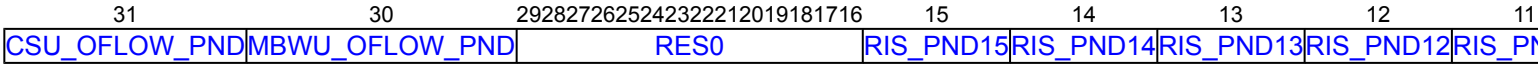
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, and MPAMF_MSMON_IDR.HAS_OFLOW_SR == 1. Otherwise, direct accesses to MSMON_OFLOW_SR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_SR is a 32-bit register.

Field descriptions



CSU_OFLOW_PND, bit [31]

At least one cache storage usage monitor has OFLOW_STATUS of 1 in MSMON_CFG_CSU_CTL.

CSU_OFLOW_PND	Meaning
0b0	There are no cache storage usage monitor instances where MSMON_CFG_CSU_CTL.OFLOW_STATUS is 1.
0b1	MSMON_CFG_CSU_CTL for at least one of the cache storage usage monitor instances has OFLOW_STATUS set to 1.

This field clears when MSMON_CFG_CSU_CTL.OFLOW_STATUS has been reset to 0 for all CSU monitor instances in this MSC.

MBWU_OFLOW_PND, bit [30]

At least one memory bandwidth usage monitor instance has OFLOW_STATUS or OFLOW_STATUS_L of 1 in MSMON_CFG_MBWU_CTL.

MBWU_OFLOW_PND	Meaning
0b0	There are no memory bandwidth usage monitor instances where MSMON_CFG_MBWU_CTL.OFLOW_STATUS is 1.
0b1	MSMON_CFG_MBWU_CTL for at least one of the memory bandwidth usage monitor instances has either OFLOW_STATUS or OFLOW_STATUS_L set to 1.

This field clears when MSMON_CFG_MBWU_CTL.OFLOW_STATUS and MSMON_CFG_MBWU_CTL.OFLOW_STATUS_L have been reset to 0 for all MBWU monitor instances in this MSC.

Bits [29:16]

Reserved, RES0.

RIS_PND<r>, bit [r], for r = 15 to 0

Overflow status by RIS.

RIS_PND<r>	Meaning
0b0	RIS r has no unread overflows of any type of monitor.
0b1	RIS r has at least one unread overflow in at least one of the monitor types.

Combined with the CSU_OFLOW_PND and MBWU_OFLOW_PND flags in this register, an interrupt service routine could poll only the monitor types indicated in monitors for the resource instances flagged in this field.

Bit r is set when any monitor instance of any type in resource instance r has OFLOW_STATUS or OFLOW_STATUS_L set to 1.

Accessing MSMON_OFLOW_SR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLOW_SR_s must only be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_SR_ns must only be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_SR_rt must only be accessible from the Root MPAM feature page.
- MSMON_OFLOW_SR_rl must only be accessible from the Realm MPAM feature page.

MSMON_OFLOW_SR_s, MSMON_OFLOW_SR_ns, MSMON_OFLOW_SR_rt, and MSMON_OFLOW_SR_rl must be separate registers:

- The Secure instance (MSMON_OFLOW_SR_s) accesses the monitor overflow status summary of Secure monitors.
- The Non-secure instance (MSMON_OFLOW_SR_ns) accesses the monitor overflow status summary of Non-secure monitors.
- The Root instance (MSMON_OFLOW_SR_rt) accesses the monitor overflow status summary of Root monitors.
- The Realm instance (MSMON_OFLOW_SR_rl) accesses the monitor overflow status summary of Realm monitors.

MSMON_OFLOW_SR can be accessed through the memory-mapped interfaces:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x08F0	MSMON_OFLOW_SR_s

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x08F0	MSMON_OFLOW_SR_ns

Accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x08F0	MSMON_OFLOW_SR_rt

When FEAT_RME is implemented, accesses to this register are **RO**.

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08F0	MSMON_OFLOW_SR_rl

When FEAT_RME is implemented, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSLAR_EL1, OS Lock Access Register

The OSLAR_EL1 characteristics are:

Purpose

Used to lock or unlock the OS Lock.

Configuration

External register OSLAR_EL1 bits [31:0] are architecturally mapped to AArch64 System register OSLAR_EL1[31:0].

OSLAR_EL1 is in the Core power domain.

The OS Lock can also be locked or unlocked using DBGOSLAR.

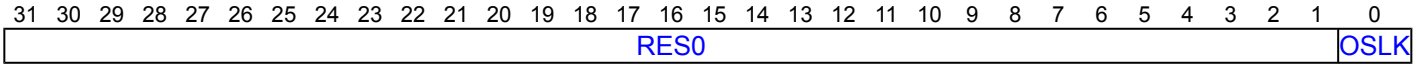
If FEAT_Debugv8p2 is not implemented, it is IMPLEMENTATION DEFINED whether external debug accesses to OSLAR_EL1 are ignored and return an error when AllowExternalDebugAccess() returns FALSE for the access.

If FEAT_Debugv8p2 is implemented, external debug accesses to OSLAR_EL1 are ignored and return an error when AllowExternalDebugAccess() returns FALSE for the access.

Attributes

OSLAR_EL1 is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

OSLK, bit [0]

On writes to OSLAR_EL1, bit[0] is copied to the OS Lock.

Use EDPRSR.OSLK to check the current status of the lock.

Accessing OSLAR_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

OSLAR_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
Debug	0x300	OSLAR_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or (!AllowExternalDebugAccess(addrdesc) and FEAT_Debugv8p2 is implemented), accesses to this register generate an error response.
- When AllowExternalDebugAccess(addrdesc) and SoftwareLockStatus(), accesses to this register are **WI**.
- When AllowExternalDebugAccess(addrdesc) and !SoftwareLockStatus(), accesses to this register are **WO**.
- Otherwise, accesses to this register are **IMPDEF**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBAUTHSTATUS, Authentication Status Register

The TRBAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

For additional information, see the CoreSight Architecture Specification.

Configuration

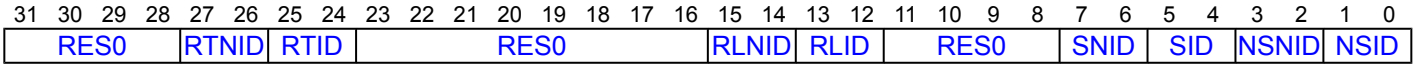
TRBAUTHSTATUS is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBAUTHSTATUS are RES0.

Attributes

TRBAUTHSTATUS is a 32-bit register.

Field descriptions



Bits [31:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

RTNID	Meaning
0b00	Not implemented.

RTID, bits [25:24]

Root invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTID.

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

RLNID	Meaning
0b00	Not implemented.

RLID, bits [13:12]

Realm invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLID.

Bits [11:8]

Reserved, RES0.

SNID, bits [7:6]

Secure non-invasive debug.

SNID	Meaning
0b00	Not implemented.

SID, bits [5:4]

Secure invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.SID.

NSNID, bits [3:2]

Non-secure non-invasive debug.

NSNID	Meaning
0b00	Not implemented.

NSID, bits [1:0]

Non-secure invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.NSID.

Accessing TRBAUTHSTATUS

TRBAUTHSTATUS can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFB8	TRBAUTHSTATUS

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBBASER_EL1, Trace Buffer Base Address Register

The TRBBASER_EL1 characteristics are:

Purpose

Defines the base address for the trace buffer.

Configuration

External register TRBBASER_EL1 bits [63:0] are architecturally mapped to AArch64 System register TRBBASER_EL1[63:0].

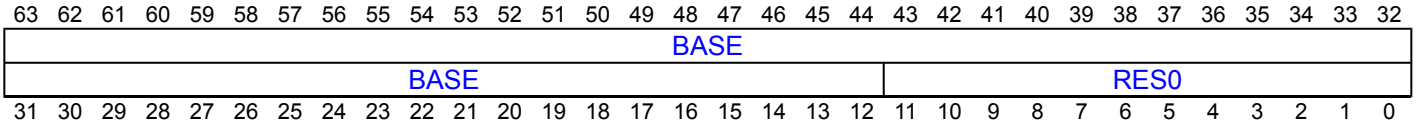
TRBBASER_EL1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBBASER_EL1 are RES0.

Attributes

TRBBASER_EL1 is a 64-bit register.

Field descriptions



BASE, bits [63:12]

Trace Buffer Base pointer address. (TRBBASER_EL1.BASE << 12) is the address of the first byte in the trace buffer. Bits [11:0] of the Base pointer address are always zero. If the smallest implemented translation granule is not 4KB, then TRBBASER_EL1[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value Log_2(smallest implemented translation granule).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

Accessing TRBBASER_EL1

The PE might ignore a write to TRBBASER_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1 and the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1 and the Trace Buffer Unit is using External mode.

TRBBASER_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0x000	TRBBASER_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalTraceBufferAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBCIDR0, Component Identification Register 0

The TRBCIDR0 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

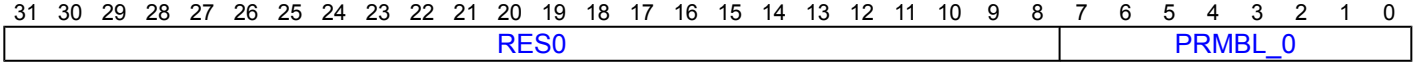
TRBCIDR0 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBCIDR0 are RES0.

Attributes

TRBCIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Component identification preamble, segment 0.

Reads as `0x0D`.

Access to this field is **RO**.

Accessing TRBCIDR0

TRBCIDR0 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	`0xFF0`	TRBCIDR0

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBCIDR1, Component Identification Register 1

The TRBCIDR1 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

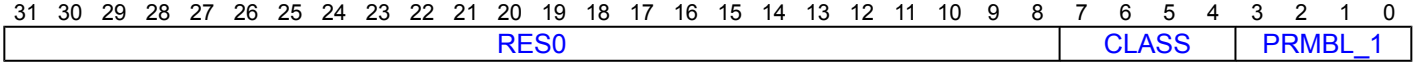
TRBCIDR1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBCIDR1 are RES0.

Attributes

TRBCIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

CLASS	Meaning
0b1001	CoreSight peripheral.

Other values are defined by the CoreSight Architecture.

Access to this field is **RO**.

PRMBL_1, bits [3:0]

Component identification preamble, segment 1.

Reads as 0b0000.

Access to this field is **RO**.

Accessing TRBCIDR1

TRBCIDR1 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFF4	TRBCIDR1

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBCIDR2, Component Identification Register 2

The TRBCIDR2 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

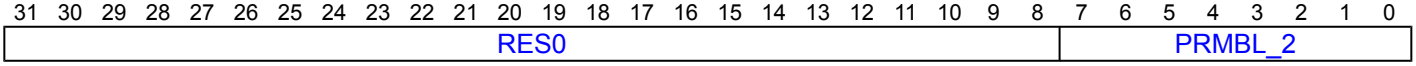
TRBCIDR2 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBCIDR2 are RES0.

Attributes

TRBCIDR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Component identification preamble, segment 2.

Reads as `0x05`.

Access to this field is **RO**.

Accessing TRBCIDR2

TRBCIDR2 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	`0xFF8`	TRBCIDR2

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBCIDR3, Component Identification Register 3

The TRBCIDR3 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

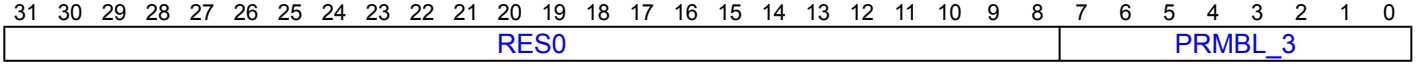
TRBCIDR3 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBCIDR3 are RES0.

Attributes

TRBCIDR3 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Component identification preamble, segment 3.

Reads as `0xB1`.

Access to this field is **RO**.

Accessing TRBCIDR3

TRBCIDR3 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	`0xFFC`	TRBCIDR3

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBCR, Trace Buffer Control Register

The TRBCR characteristics are:

Purpose

Provides trace buffer controls for an external debugger.

Configuration

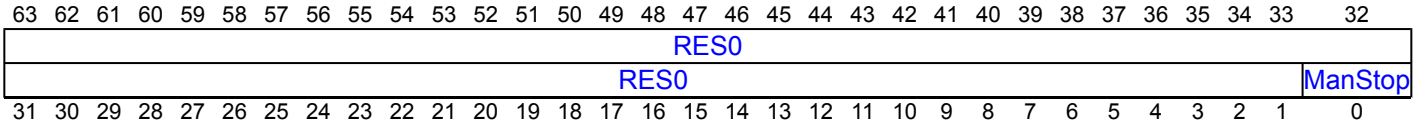
TRBCR is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBCR are RES0.

Attributes

TRBCR is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

ManStop, bit [0]

Flush and Stop collection. A write of 1 to this field causes a trace buffer flush, and on completion of the flush, Collection is stopped and the Trace Buffer Unit writes all trace data it has Accepted from the trace unit to memory, adding padding data if necessary.

Access to this field is **WO/RAZ**.

Accessing TRBCR

TRBCR can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0x038	TRBCR

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalTraceBufferAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBDEVAFF, Device Affinity Register

The TRBDEVAFF characteristics are:

Purpose

For additional information, see the CoreSight Architecture Specification.

Reads the same value as the MPIDR_EL1 register for the PE that this trace buffer has affinity with.

Depending on the IMPLEMENTATION DEFINED nature of the system, it might be possible that TRBDEVAFF is read before system firmware has configured the trace buffer and/or the PE or group of PEs that the trace buffer has affinity with. When this is the case, TRBDEVAFF reads as zero.

Configuration

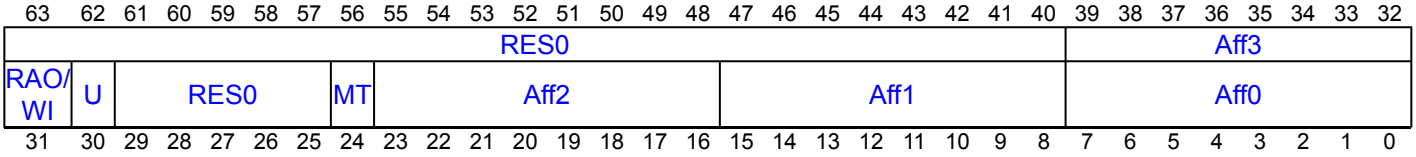
TRBDEVAFF is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBDEVAFF are RES0.

Attributes

TRBDEVAFF is a 64-bit register.

Field descriptions



Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [31]

Reserved, RAO/WI.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Access to this field is **RO**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is **RO**.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRBDEVAFF

TRBDEVAFF can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFA8	TRBDEVAFF

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.

- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBDEVARCH, Trace Buffer Device Architecture Register

The TRBDEVARCH characteristics are:

Purpose

Provides discovery information for the component.

Configuration

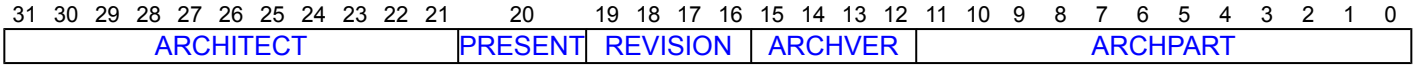
TRBDEVARCH is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBDEVARCH are RES0.

Attributes

TRBDEVARCH is a 32-bit register.

Field descriptions



ARCHITECT, bits [31:21]

Defines the architect of the component. For Trace Buffer, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, `0b0100`.

Bits [27:21] are the JEP106 identification code, `0b0111011`.

Reads as `0b01000111011`.

Access to this field is **RO**.

PRESENT, bit [20]

DEVARCH present. Indicates that the TRBDEVARCH register is present.

Reads as `0b1`.

Access to this field is **RO**.

REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

REVISION	Meaning
`0b0000`	First revision.
`0b0001`	As `0b0000`, and adds: • If EL2 and FEAT_FGT are implemented, a fine-grained trap on the `TSB CSYNC` instruction. • If EL2 is implemented, an EL2 control to override TRBLIMITR_EL1.nVM. • The TRBE Profiling exception extension, FEAT_TRBE_EXC.

All other values are reserved.

FEAT_TRBE implements the functionality identified by the value 0b0000.

FEAT_TRBEv1p1 implements the functionality identified by the value 0b0001.

From Armv9.6, the value 0b0000 is not permitted.

Access to this field is **RO**.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

ARCHVER	Meaning
0b0000	Trace Buffer Extension version 1.

All other values are reserved.

TRBDEVARCH.ARCHVER and TRBDEVARCH.ARCHPART are also defined as a single field, TRBDEVARCH.ARCHID, so that TRBDEVARCH.ARCHVER is TRBDEVARCH.ARCHID[15:12].

Access to this field is **RO**.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

ARCHPART	Meaning
0xA18	Armv9-A Trace Buffer Extension.

TRBDEVARCH.ARCHVER and TRBDEVARCH.ARCHPART are also defined as a single field, TRBDEVARCH.ARCHID, so that TRBDEVARCH.ARCHPART is TRBDEVARCH.ARCHID[11:0].

Access to this field is **RO**.

Accessing TRBDEVARCH

TRBDEVARCH can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFBC	TRBDEVARCH

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBDEVID, Device Configuration Register

The TRBDEVID characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

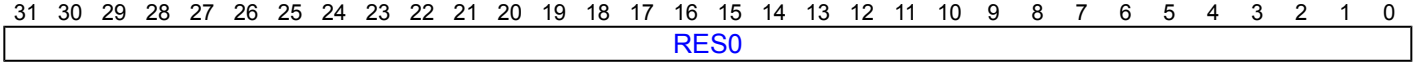
TRBDEVID is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBDEVID are RES0.

Attributes

TRBDEVID is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRBDEVID

TRBDEVID can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFC8	TRBDEVID

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBDEVID1, Device Configuration Register 1

The TRBDEVID1 characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

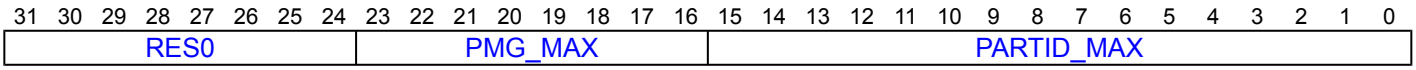
TRBDEVID1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBDEVID1 are RES0.

Attributes

TRBDEVID1 is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

PMG_MAX, bits [23:16]

When FEAT_TRBE_MPAM is implemented:

Largest permitted PMG value. The TRBMPAM_EL1.PMG field must implement at least enough bits to represent TRBDEVID1.PMG_MAX.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

PARTID_MAX, bits [15:0]

When FEAT_TRBE_MPAM is implemented:

Largest permitted PARTID value. The TRBMPAM_EL1.PARTID field must implement at least enough bits to represent TRBDEVID1.PARTID_MAX.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRBDEVID1

TRBDEVID1 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFC4	TRBDEVID1

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBDEVID2, Device Configuration Register 2

The TRBDEVID2 characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

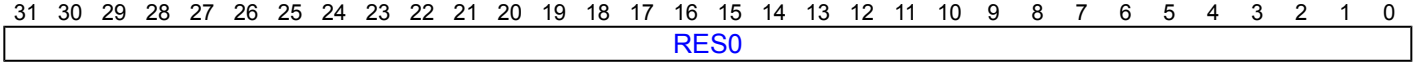
TRBDEVID2 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBDEVID2 are RES0.

Attributes

TRBDEVID2 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRBDEVID2

TRBDEVID2 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFC0	TRBDEVID2

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBDEVTYPE, Device Type Register

The TRBDEVTYPE characteristics are:

Purpose

Provides discovery information for the component. If the part number field is not recognized, a debugger can report the information that is provided by TRBDEVTYPE about the component instead.

For additional information, see the CoreSight Architecture Specification.

Configuration

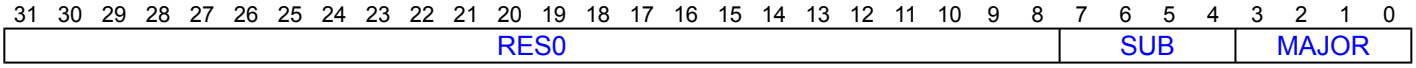
TRBDEVTYPE is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBDEVTYPE are RES0.

Attributes

TRBDEVTYPE is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Component sub-type.

SUB	Meaning
0b0010	When MAJOR == 0x1 (Trace sink), Trace buffer or router.

This field reads as 0x2.

Access to this field is **RO**.

MAJOR, bits [3:0]

Component major type.

MAJOR	Meaning
0b0001	Trace sink.

This field reads as 0x1.

Access to this field is **RO**.

Accessing TRBDEVTYPE

TRBDEVTYPE can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFCC	TRBDEVTYPE

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBIDR_EL1, Trace Buffer ID Register

The TRBIDR_EL1 characteristics are:

Purpose

Describes constraints on using the Trace Buffer Unit to an external debugger.

Configuration

External register TRBIDR_EL1 bits [63:0] are architecturally mapped to AArch64 System register TRBIDR_EL1[63:0].

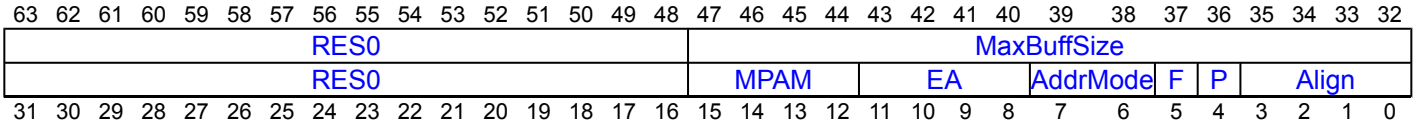
TRBIDR_EL1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBIDR_EL1 are RES0.

Attributes

TRBIDR_EL1 is a 64-bit register.

Field descriptions



Bits [63:48]

Reserved, RES0.

MaxBuffSize, bits [47:32]

Maximum supported trace buffer size. Reserved for software use.

The only permitted value is `0x0000`, indicating there is no limit to the maximum buffer size.

Reads as `0x0000`.

Access to this field is **RO**.

Bits [31:16]

Reserved, RES0.

MPAM, bits [15:12]

When FEAT_TRBE_EXT is implemented:

MPAM extensions. Indicates Memory Partitioning and Monitoring (MPAM) support in the Trace Buffer Unit when using External mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MPAM	Meaning
0b0000	Trace Buffer External Mode is not implemented or MPAM is not implemented by the PE.
0b0001	MPAM implemented by the Trace Buffer Unit, using default PARTID and PMG values in External mode.
0b0010	Trace Buffer MPAM extensions implemented.

When FEAT_MPAM is not implemented by the PE, the only permitted value is 0b0000.

When FEAT_MPAM is implemented by the PE, the value 0b0000 is not permitted.

FEAT_TRBE_MPAM implements the functionality identified by the value 0b0010.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

EA, bits [11:8]

External Abort handling. Describes how the PE manages External aborts on writes made by the Trace Buffer Unit to the trace buffer.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EA	Meaning
0b0000	Not described.
0b0001	The PE ignores External aborts on writes made by the Trace Buffer Unit.
0b0010	An External abort on a write made by the Trace Buffer Unit generates an asynchronous SError exception at the PE.

All other values are reserved.

From Armv9.3, the value 0b0000 is not permitted.

TRBIDR_EL1.EA describes only External aborts generated by the write to memory. External aborts on a translation table walk made by the Trace Buffer Unit generate trace buffer management events reported as MMU faults using TRBSR_ELx.

Access to this field is **RO**.

AddrMode, bits [7:6]

This field reads as an UNKNOWN value.

F, bit [5]

Flag updates. Describes how address translations performed by the Trace Buffer Unit manage the Access flag and dirty state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F	Meaning
0b0	Hardware management of the Access flag and dirty state for accesses made by the Trace Buffer Unit is always disabled for all translation stages.
0b1	Hardware management of the Access flag and dirty state for accesses made by the Trace Buffer Unit is controlled in the same way as explicit memory accesses in the trace buffer owning translation regime.

Note

If hardware management of the Access flag is disabled for a stage of translation, an access to a Page or Block with the Access flag bit not set in the descriptor will generate an Access Flag fault.

If hardware management of the dirty state is disabled for a stage of translation, an access to a Page or Block will ignore the Dirty Bit Modifier in the descriptor and might generate a Permission fault, depending on the values of the access permission bits in the descriptor.

From Armv9.3, the value 0 is not permitted.

Access to this field is **RO**.

P, bit [4]

This field reads as an UNKNOWN value.

Align, bits [3:0]

Defines the minimum alignment constraint for writes to TRBPTR_EL1 and TRBTRG_EL1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Align	Meaning
0b0000	Byte.
0b0001	Halfword.
0b0010	Word.
0b0011	Doubleword.
0b0100	16 bytes.
0b0101	32 bytes.
0b0110	64 bytes.
0b0111	128 bytes.
0b1000	256 bytes.
0b1001	512 bytes.
0b1010	1KB.
0b1011	2KB.

All other values are reserved.

Access to this field is **RO**.

Accessing TRBIDR_EL1

TRBIDR_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0x030	TRBIDR_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalTraceBufferAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBITCTRL, Integration Mode Control Register

The TRBITCTRL characteristics are:

Purpose

A component can use TRBITCTRL to dynamically switch between functional mode and integration mode. In integration mode, topology detection is enabled. After switching to integration mode and performing integration tests or topology detection, reset the system to ensure correct behavior of CoreSight and other connected system components.

For additional information, see the CoreSight Architecture Specification.

Configuration

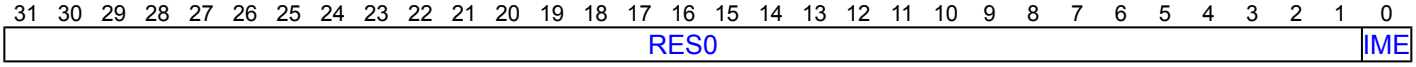
TRBITCTRL is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBITCTRL are RES0.

Attributes

TRBITCTRL is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

IME, bit [0]

When topology detection or integration functionality is implemented:

Integration Mode Enable.

IME	Meaning
0b0	Component functional mode.
0b1	Component integration mode. Support for topology detection and integration testing is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRBITCTRL

The PE might ignore a write to TRBITCTRL if any of the following apply:

- TRBLIMITR_EL1.E == 1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

TRBITCTRL can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xF00	TRBITCTRL

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalTraceBufferAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBLAR, Lock Access Register

The TRBLAR characteristics are:

Purpose

For components that implement the Software Lock, used to lock and unlock the Software Lock. This component does not implement the Software Lock.

For additional information, see the CoreSight Architecture Specification.

Configuration

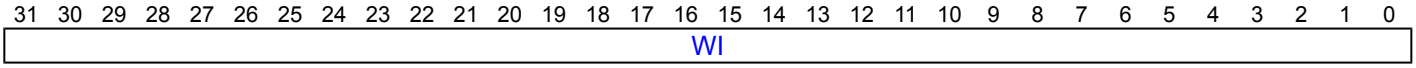
TRBLAR is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBLAR are RES0.

Attributes

TRBLAR is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, WI.

Software Lock Key. The Software Lock is not implemented.

This field ignores writes.

Accessing TRBLAR

TRBLAR can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFB0	TRBLAR

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBLIMITR_EL1, Trace Buffer Limit Address Register

The TRBLIMITR_EL1 characteristics are:

Purpose

Defines the top address for the trace buffer, and controls the trace buffer modes and enable.

Configuration

External register TRBLIMITR_EL1 bits [63:0] are architecturally mapped to AArch64 System register TRBLIMITR_EL1[63:0].

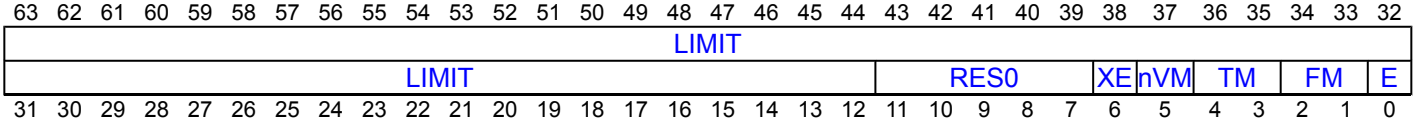
TRBLIMITR_EL1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBLIMITR_EL1 are RES0.

Attributes

TRBLIMITR_EL1 is a 64-bit register.

Field descriptions



LIMIT, bits [63:12]

Trace buffer Limit pointer address. (TRBLIMITR_EL1.LIMIT << 12) is the address of the last byte in the trace buffer plus one. Bits [11:0] of the Limit pointer address are always zero. If the smallest implemented translation granule is not 4KB, then TRBLIMITR_EL1[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value Log_2(smallest implemented translation granule).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [11:7]

Reserved, RES0.

XE, bit [6]

Trace Buffer Unit External mode enable. Controls whether the Trace Buffer Unit is enabled when SelfHostedTraceEnabled() == FALSE.

XE	Meaning
0b0	Trace Buffer Unit is not enabled by this control.
0b1	If SelfHostedTraceEnabled() is FALSE, the Trace Buffer Unit is enabled.

If SelfHostedTraceEnabled() == TRUE, then TRBLIMITR_EL1.E controls whether the Trace Buffer Unit is enabled.

All output is discarded by the Trace Buffer Unit when the Trace Buffer Unit is disabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

nVM, bit [5]

Address mode.

nVM	Meaning
0b0	The trace buffer pointers are virtual addresses.
0b1	The trace buffer pointers are: • Physical address in the owning security state if the owning translation regime has no stage 2 translation. • Intermediate physical addresses in the owning security state if the owning translation regime has stage 2 translations.

If SelfHostedTraceEnabled() == FALSE, then the PE ignores the value of this field and the trace buffer pointers are always physical addresses.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When !SelfHostedTraceEnabled(), access to this field is **RES1**.
- Otherwise, access to this field is **RW**.

TM, bits [4:3]

Trigger mode.

TM	Meaning
0b00	Stop on trigger. Flush trace, then stop collection and set TRBSR_EL1.IRQ to 1 on Trigger Event.
0b01	IRQ on trigger. Continue collection and set TRBSR_EL1.IRQ to 1 on Trigger Event.
0b11	Ignore trigger. Continue collection and leave TRBSR_EL1.IRQ unchanged on Trigger Event.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

FM, bits [2:1]

Trace buffer mode.

FM	Meaning
0b00	Fill mode. Stop collection and set TRBSR_EL1.IRQ to 1 on current write pointer wrap.
0b01	Wrap mode. Continue collection and set TRBSR_EL1.IRQ to 1 on current write pointer wrap.
0b11	Circular Buffer mode. Continue collection and leave TRBSR_EL1.IRQ unchanged on current write pointer wrap.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Trace Buffer Unit enable. Controls whether the Trace Buffer Unit is enabled when SelfHostedTraceEnabled() == TRUE.

E	Meaning
0b0	Trace Buffer Unit is not enabled by this control.
0b1	If SelfHostedTraceEnabled() is TRUE, the Trace Buffer Unit is enabled.

If FEAT_TRBE_EXT is implemented and SelfHostedTraceEnabled() == FALSE, then TRBLIMITR_EL1.XE controls whether the Trace Buffer Unit is enabled.

If FEAT_TRBE_EXT is not implemented, then the Trace Buffer Unit is disabled when SelfHostedTraceEnabled() == FALSE.

All output is discarded by the Trace Buffer Unit when the Trace Buffer Unit is disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing TRBLIMITR_EL1

The PE might ignore a write to TRBLIMITR_EL1, other than a write that modifies TRBLIMITR_EL1.E or TRBLIMITR_EL1.XE as appropriate, if any of the following apply:

- TRBLIMITR_EL1.E == `0b1`, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == `0b1`, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

TRBLIMITR_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	`0x010`	TRBLIMITR_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalTraceBufferAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBLSR, Lock Status Register

The TRBLSR characteristics are:

Purpose

Indicates the Software Lock is not implemented.

For additional information, see the CoreSight Architecture Specification.

Configuration

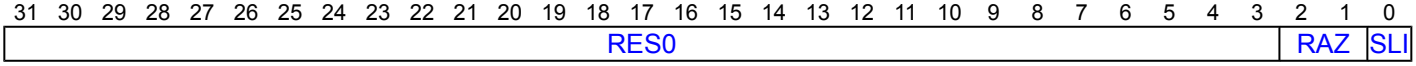
TRBLSR is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBLSR are RES0.

Attributes

TRBLSR is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

Bits [2:1]

Reserved, RAZ.

Not thirty-two bit. Describes the size of the TRBLAR register.

This field reads-as-zero.

SLI, bit [0]

Indicates the Software Lock is not implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SLI	Meaning
0b0	Software Lock is not implemented. Writes to the TRBLAR are ignored.
0b1	Software Lock is implemented.

Access to this field is **RAZ/WI**.

Accessing TRBLSR

TRBLSR can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFB4	TRBLSR

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBMAR_EL1, Trace Buffer Memory Attribute Register

The TRBMAR_EL1 characteristics are:

Purpose

Controls Trace Buffer Unit accesses to memory.

Configuration

External register TRBMAR_EL1 bits [63:0] are architecturally mapped to AArch64 System register TRBMAR_EL1[63:0].

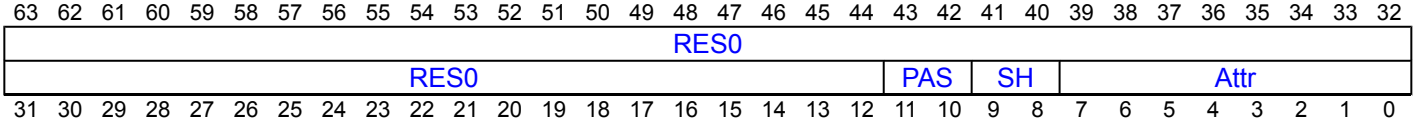
TRBMAR_EL1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBMAR_EL1 are RES0.

Attributes

TRBMAR_EL1 is a 64-bit register.

Field descriptions



Bits [63:12]

Reserved, RES0.

PAS, bits [11:10]

Physical address specifier. Defines the PAS attribute for memory addressed by the buffer in External mode.

PAS	Meaning	Applies when
0b00	Secure.	When Secure state is implemented
0b01	Non-secure.	
0b10	Root.	When FEAT_RME is implemented
0b11	Realm.	When FEAT_RME is implemented

All other values are reserved.

If the Trace Buffer Unit is using external mode and either TRBMAR_EL1.PAS is set to a reserved value, or the IMPLEMENTATION DEFINED authentication interface prohibits invasive debug of the Security state corresponding to the physical address space selected by TRBMAR_EL1.PAS, then when the Trace Buffer Unit receives trace data from the trace unit, it does not write the trace data to memory and generates a trace buffer management event. That is, if any of the following apply:

- `ExternalInvasiveDebugEnabled()` == FALSE.
- TRBMAR_EL1.PAS is `0b00`, and either `ExternalSecureInvasiveDebugEnabled()` == FALSE or Secure state is not implemented.
- TRBMAR_EL1.PAS is `0b10`, and either `ExternalRootInvasiveDebugEnabled()` == FALSE or FEAT_RME is not implemented.
- TRBMAR_EL1.PAS is `0b11`, and either `ExternalRealmInvasiveDebugEnabled()` == FALSE or FEAT_RME is not implemented.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SH, bits [9:8]

Trace buffer shareability domain. Defines the shareability domain for Normal memory used by the trace buffer.

SH	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when TRBMAR_EL1.Attr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Attr, bits [7:0]

Trace buffer memory type and attributes. Defines the memory type and, for Normal memory, the cacheability attributes, for memory addressed by the trace buffer.

The encoding of this field is the same as that of a MAIR_ELx.Attr<n> field, as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of 'dd' for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise,UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise,UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBMAR_EL1

The PE might ignore a write to TRBMAR_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1 and the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1 and the Trace Buffer Unit is using External mode.

TRBMAR_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0x028	TRBMAR_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalTraceBufferAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBMPAM_EL1, Trace Buffer MPAM Configuration Register

The TRBMPAM_EL1 characteristics are:

Purpose

Defines the PARTID, PMG, and MPAM_SP values used by the trace buffer unit in external mode.

Configuration

External register TRBMPAM_EL1 bits [63:0] are architecturally mapped to AArch64 System register TRBMPAM_EL1[63:0].

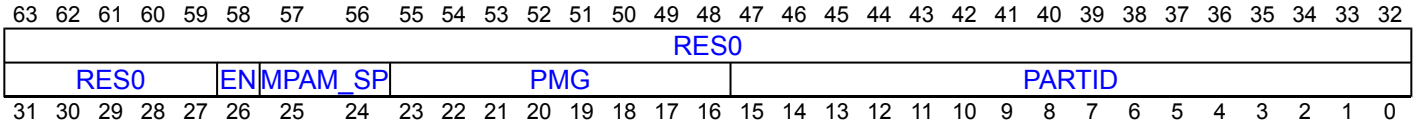
TRBMPAM_EL1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented and FEAT_TRBE_MPAM is implemented. Otherwise, direct accesses to TRBMPAM_EL1 are RES0.

Attributes

TRBMPAM_EL1 is a 64-bit register.

Field descriptions



Bits [63:27]

Reserved, RES0.

EN, bit [26]

Enable. Enables use of non-default MPAM values.

EN	Meaning
0b0	Use default MPAM values.
0b1	Use TRBMPAM_EL1.{PARTID, PMG, MPAM_SP}.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

MPAM_SP, bits [25:24]

Partition Identifier space. Selects the PARTID space.

MPAM_SP	Meaning	Applies when
0b00	PARTID is in the Secure PARTID space.	When Secure state is implemented
0b01	PARTID is in the Non-secure PARTID space.	
0b10	PARTID is in the Root PARTID space.	When FEAT_RME is implemented
0b11	PARTID is in the Realm PARTID space.	When FEAT_RME is implemented

All other values are reserved.

If the Trace Buffer Unit is using external mode and either TRBMPAM_EL1.MPAM_SP is set to reserved value, or the IMPLEMENTATION DEFINED authentication interface prohibits invasive debug of the Security state corresponding to the Partition Identifier space selected by TRBMPAM_EL1.MPAM_SP, then when the Trace Buffer Unit receives trace data from the trace unit, it does not write the trace data to memory and generates a trace buffer management event. That is, if any of the following apply:

- `ExternalInvasiveDebugEnabled()` == FALSE.
- TRBMPAM_EL1.MPAM_SP is `0b00`, and either `ExternalSecureInvasiveDebugEnabled()` == FALSE or Secure state is not implemented.
- TRBMPAM_EL1.MPAM_SP is `0b10`, and either `ExternalRootInvasiveDebugEnabled()` == FALSE or FEAT_RME is not implemented.
- TRBMPAM_EL1.MPAM_SP is `0b11`, and either `ExternalRealmInvasiveDebugEnabled()` == FALSE or FEAT_RME is not implemented.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PMG, bits [23:16]

Performance Monitoring Group. Selects the PMG.

Only sufficient low-order bits are required to represent the TRBDEVID1.PMG_MAX. Higher-order bits are RES0.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PARTID, bits [15:0]

Partition Identifier. Selects the PARTID.

Only sufficient low-order bits are required to represent the TRBDEVID1.PARTID_MAX. Higher-order bits are RES0.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBMPAM_EL1

The PE might ignore a write to TRBMPAM_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == `0b1` and the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == `0b1` and the Trace Buffer Unit is using External mode.

TRBMPAM_EL1 can be accessed through the external debug interface:

Component	Offset	Instance

TRBE	0x040	TRBMPAM_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalTraceBufferAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPIDR0, Peripheral Identification Register 0

The TRBPIDR0 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

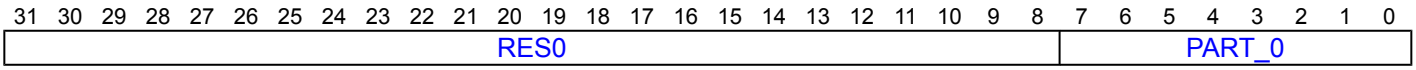
TRBPIDR0 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBPIDR0 are RES0.

Attributes

TRBPIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, bits [7:0].

The part number is selected by the designer of the component, and is stored in TRBPIDR1.PART_1 and TRBPIDR0.PART_0.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRBPIDR0

TRBPIDR0 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFE0	TRBPIDR0

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPIDR1, Peripheral Identification Register 1

The TRBPIDR1 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

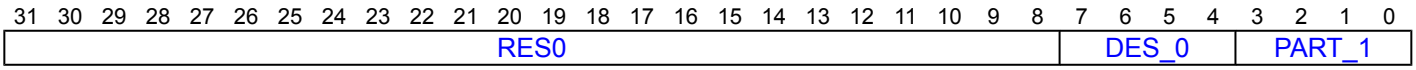
TRBPIDR1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBPIDR1 are RES0.

Attributes

TRBPIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, JEP106 identification code, bits [3:0]. TRBPIDR1.DES_0 and TRBPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

Note

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, bits [11:8].

The part number is selected by the designer of the component, and is stored in TRBPIDR1.PART_1 and TRBPIDR0.PART_0.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRBPIDR1

TRBPIDR1 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFE4	TRBPIDR1

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPIDR2, Peripheral Identification Register 2

The TRBPIDR2 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

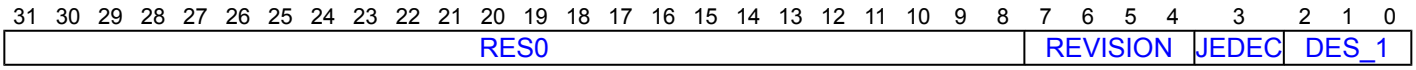
TRBPIDR2 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBPIDR2 are RES0.

Attributes

TRBPIDR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Component major revision. TRBPIDR2.REVISION and TRBPIDR3.REVAND together form the revision number of the component, with TRBPIDR2.REVISION being the most significant part and TRBPIDR3.REVAND the least significant part. When a component is changed, TRBPIDR2.REVISION or TRBPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. TRBPIDR3.REVAND should be set to `0b0000` when TRBPIDR2.REVISION is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as `0b1`.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, JEP106 identification code, bits [6:4]. TRBPIDR1.DES_0 and TRBPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

Note

For a component designed by Arm Limited, the JEP106 identification code is `0x3B`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRBPIDR2

TRBPIDR2 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	`0xFE8`	TRBPIDR2

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPIDR3, Peripheral Identification Register 3

The TRBPIDR3 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

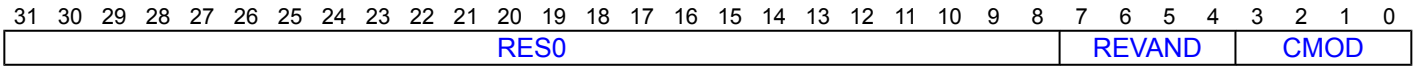
TRBPIDR3 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBPIDR3 are RES0.

Attributes

TRBPIDR3 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Component minor revision. TRBPIDR2.REVISION and TRBPIDR3.REVAND together form the revision number of the component, with TRBPIDR2.REVISION being the most significant part and TRBPIDR3.REVAND the least significant part. When a component is changed, TRBPIDR2.REVISION or TRBPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. TRBPIDR3.REVAND should be set to `0b0000` when TRBPIDR2.REVISION is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer Modified.

Indicates the component has been modified.

A value of `0b0000` means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If TRBPIDR3.CMOD is zero in both components, then the components are identical.
- If TRBPIDR3.CMOD has the same nonzero value in both components, then this does not necessarily mean that they have the same modifications.
- If TRBPIDR3.CMOD is nonzero in either component, the two components might not be identical despite having the same Unique Component Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRBPIDR3

TRBPIDR3 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFEC	TRBPIDR3

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPIDR4, Peripheral Identification Register 4

The TRBPIDR4 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

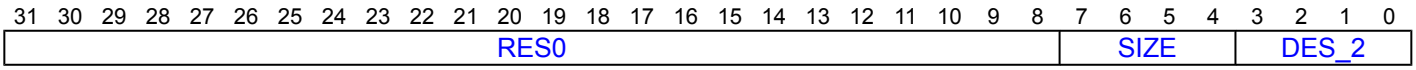
TRBPIDR4 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBPIDR4 are RES0.

Attributes

TRBPIDR4 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component.

The distance from the start of the address space used by this component to the end of the component identification registers.

A value of `0b0000` means one of the following is true:

- The component uses a single 4KB block.
- The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.

Any other value means the component occupies $2^{\text{TRBPIDR4.SIZE}}$ 4KB blocks.

Using this field to indicate the size of the component is deprecated. This field might not correctly indicate the size of the component. Arm recommends that software determine the size of the component from the Unique Component Identifier fields, and other IMPLEMENTATION DEFINED registers in the component.

Reads as `0b0000`.

Access to this field is **RO**.

DES_2, bits [3:0]

Designer, JEP106 continuation code. This is the JEDEC-assigned JEP106 bank identifier for the designer of the component, minus 1. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

Note

For a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value `0x4`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRBPIDR4

TRBPIDR4 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	`0xFD0`	TRBPIDR4

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPIDR5, Peripheral Identification Register 5

The TRBPIDR5 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

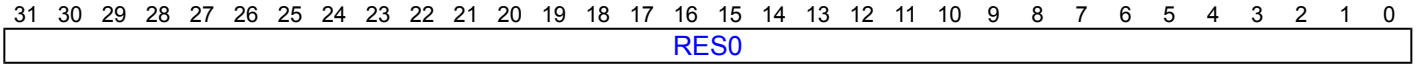
TRBPIDR5 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBPIDR5 are RES0.

Attributes

TRBPIDR5 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRBPIDR5

TRBPIDR5 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFD4	TRBPIDR5

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPIDR6, Peripheral Identification Register 6

The TRBPIDR6 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

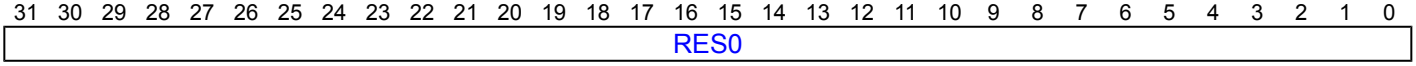
TRBPIDR6 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBPIDR6 are RES0.

Attributes

TRBPIDR6 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRBPIDR6

TRBPIDR6 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFD8	TRBPIDR6

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPIDR7, Peripheral Identification Register 7

The TRBPIDR7 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

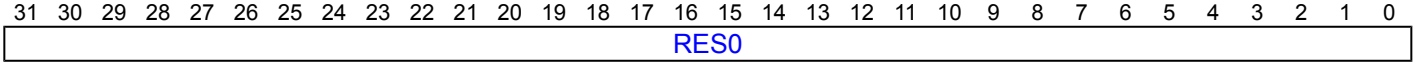
TRBPIDR7 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBPIDR7 are RES0.

Attributes

TRBPIDR7 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRBPIDR7

TRBPIDR7 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0xFDC	TRBPIDR7

Accessible as follows:

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPTR_EL1, Trace Buffer Write Pointer Register

The TRBPTR_EL1 characteristics are:

Purpose

Defines the current write pointer for the trace buffer.

Configuration

External register TRBPTR_EL1 bits [63:0] are architecturally mapped to AArch64 System register TRBPTR_EL1[63:0].

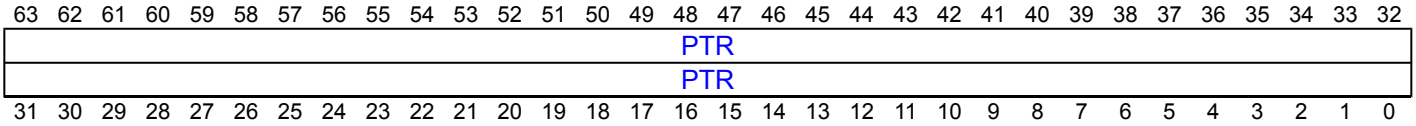
TRBPTR_EL1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBPTR_EL1 are RES0.

Attributes

TRBPTR_EL1 is a 64-bit register.

Field descriptions



PTR, bits [63:0]

Trace Buffer current write pointer address.

Defines the virtual address of the next entry to be written to the trace buffer.

If TRBIDR_EL1.Align is not zero, then it is IMPLEMENTATION DEFINED whether bits [M-1:0] are RES0 or read/write, where M is an integer between 1 and TRBIDR_EL1.Align inclusive.

The architecture places restrictions on the values that software can write to the pointer. For more information see 'Restrictions on Programming the Trace Buffer Unit'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBPTR_EL1

The PE might ignore a write to TRBPTR_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == `0b1` and the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == `0b1` and the Trace Buffer Unit is using External mode.

TRBPTR_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	`0x008`	TRBPTR_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalTraceBufferAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBSR_EL1, Trace Buffer Status/syndrome Register

The TRBSR_EL1 characteristics are:

Purpose

Provides syndrome information to software for a trace buffer management event.

Configuration

External register TRBSR_EL1 bits [63:0] are architecturally mapped to AArch64 System register TRBSR_EL1[63:0].

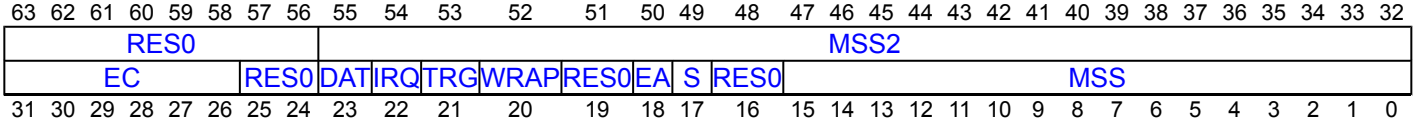
TRBSR_EL1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBSR_EL1 are RES0.

Attributes

TRBSR_EL1 is a 64-bit register.

Field descriptions



Bits [63:56]

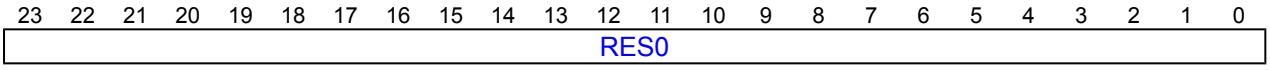
Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

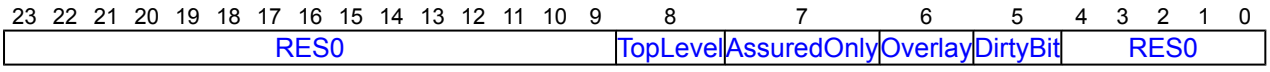
MSS2 encoding for other trace buffer management events



Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer



Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, TRBSR_EL1.EC == 0b100101, and GetTRBSR_EL1_FSC() IN {0b0011xx}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetTRBSR_EL1_FSC() IN {0b0011xx}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetTRBSR_EL1_FSC() IN {0b0011xx}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

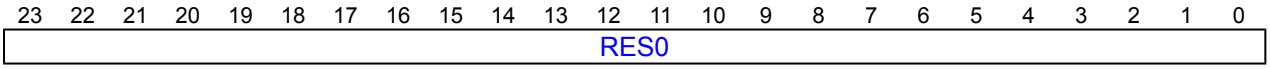
Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

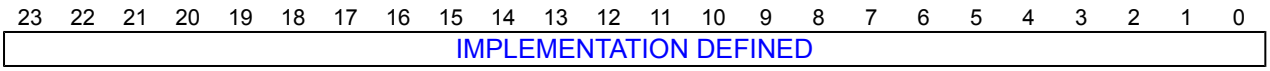
MSS2 encoding for Granule Protection Check faults on write to trace buffer



Bits [23:0]

Reserved, RES0.

MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason



IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the trace buffer management event.

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other trace buffer management event. All trace buffer management events other than those described by the other defined Event class codes.	MSS encoding for other trace buffer management events	MSS2 encoding for other trace buffer management events	
0b011110	Granule Protection Check fault on write to trace buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to trace buffer	MSS2 encoding for Granule Protection Check faults on write to trace buffer	When FEAT_RME is implemented
0b011111	Trace buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	
0b100101	Stage 2 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [25:24]

Reserved, RES0.

DAT, bit [23]

When FEAT_TRBE_EXT is implemented:

Data. Indicates when the Trace Buffer Unit has trace data that has not yet been written to memory.

DAT	Meaning
0b0	Internal buffers are empty. All Trace operations Accepted by the Trace Buffer Unit will Complete in finite time.
0b1	Internal buffers are not empty.

When TRBSR_EL1.{DAT, S} is {0, 1}, meaning Collection is stopped and the Trace Buffer Unit internal buffers are empty, then all trace data has been written to memory. An additional Data Synchronization Barrier may be required to ensure that the writes are Complete. When TRBSR_EL1.DAT is 0 and Collection is not stopped, there may still be trace data held by the trace unit that the Trace Buffer Unit has not Accepted.

That is, TRBSR_EL1.DAT reads as 1 when the Trace Buffer Unit has Accepted trace data from the trace unit, but has not yet written it to memory.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IRQ, bit [22]

Maintenance status. Indicates that a trace buffer management event has been recorded.

IRQ	Meaning
0b0	No trace buffer management event for EL1 has been recorded.
0b1	A trace buffer management event for EL1 has been recorded.

When FEAT_TRBE_EXC is implemented, this field indicates a management event for EL1.

If FEAT_TRBE_EXC is implemented and the TRBE Profiling exception for EL1 is enabled, then when this field is 1, a TRBE Profiling exception for EL1 is pending

If FEAT_TRBE_EXC is not implemented or the TRBE Profiling exception for EL1 is disabled, then this field drives the **TRBIRQ** trace buffer interrupt request signal.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

TRG, bit [21]

Triggered.

TRG	Meaning
0b0	No Detected Trigger has been observed since this field was last cleared to zero.
0b1	A Detected Trigger has been observed since this field was last cleared to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

WRAP, bit [20]

Wrapped.

WRAP	Meaning
0b0	The current write pointer has not wrapped since this field was last cleared to zero.
0b1	The current write pointer has wrapped since this field was last cleared to zero.

For each byte of trace the Trace Buffer Unit Accepts and writes to the trace buffer at the address in the current write pointer, if the current write pointer is equal to the Limit pointer minus one, the current write pointer is wrapped by setting it to the Base pointer, and this field is set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

EA, bit [18]

From Armv9.3:

Reserved, RES0.

When the PE sets this bit as the result of an External abort:

External Abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Trace Buffer Unit.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S, bit [17]

Stopped.

S	Meaning
0b0	Collection has not been stopped.
0b1	Collection is stopped.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [16]

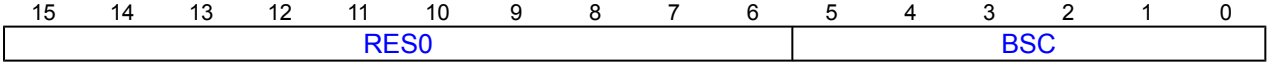
Reserved, RES0.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the trace buffer management event.

The syndrome contents for each trace buffer management event are described in the following sections.

MSS encoding for other trace buffer management events



Bits [15:6]

Reserved, RES0.

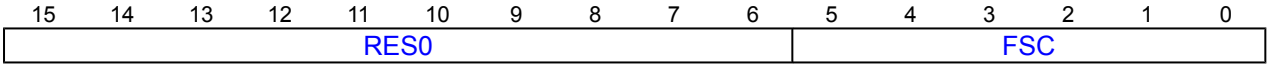
BSC, bits [5:0]

Trace buffer status code

BSC	Meaning
0b000000	Collection not stopped, or access not allowed.
0b000001	Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode.
0b000010	Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information.
0b000011	Manual Stop. Collection stopped because of a Manual Stop event. See TRBCR.ManStop for more information.
0b000100	Buffer size. The requested trace buffer size was too large.

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer



Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	When FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	When FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	When FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	When FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	When FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	When FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	When FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	When FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	When FEAT_RME is implemented

0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	When FEAT_RME is implemented
0b101001	Address size fault, level -1.	When FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	When FEAT_D128 is implemented
0b101011	Translation fault, level -1.	When FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	When FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	When FEAT_HAFDBS is implemented

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to trace buffer

Bits [15:0]

Reserved, RES0.

MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing TRBSR_EL1

The PE might ignore a write to TRBSR_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1 and the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1 and the Trace Buffer Unit is using External mode.

TRBSR_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0x018	TRBSR_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalTraceBufferAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBTRG_EL1, Trace Buffer Trigger Counter Register

The TRBTRG_EL1 characteristics are:

Purpose

Specifies the number of bytes of trace to capture following a Detected Trigger before a Trigger Event.

Configuration

External register TRBTRG_EL1 bits [63:0] are architecturally mapped to AArch64 System register TRBTRG_EL1[63:0].

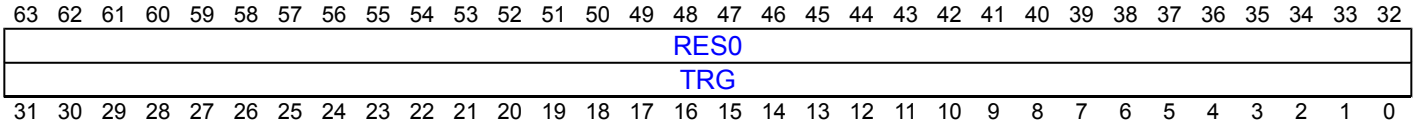
TRBTRG_EL1 is in the Core power domain.

This register is present only when FEAT_TRBE_EXT is implemented. Otherwise, direct accesses to TRBTRG_EL1 are RES0.

Attributes

TRBTRG_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TRG, bits [31:0]

Trigger count.

Specifies the number of bytes of trace to capture following a Detected Trigger before a Trigger Event.

TRBTRG_EL1 decrements by 1 for every byte of trace written to the trace buffer when all of the following are true:

- TRBTRG_EL1 is nonzero.
- TRBSR_EL1.TRG is 1.

The architecture places restrictions on the values that software can write to the counter.

Note

As a result of the restrictions an implementation might treat some of TRG[M:0] as RES0, where M is defined by TRBIDR_EL1.Align.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBTRG_EL1

The PE might ignore a write to TRBTRG_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1 and the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1 and the Trace Buffer Unit is using External mode.

TRBTRG_EL1 can be accessed through the external debug interface:

Component	Offset	Instance
TRBE	0x020	TRBTRG_EL1

Accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus(), or !AllowExternalTraceBufferAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCACATR<n>, Trace Address Comparator Access Type Register <n>, n = 0 - 15

The TRCACATR<n> characteristics are:

Purpose

Defines the type of access for the corresponding TRCACVR<n> Register. This register configures the context type, Exception levels, alignment, masking that is applied by the Address Comparator, and how the Address Comparator behaves when it is one half of an Address Range Comparator.

Configuration

External register TRCACATR<n> bits [63:0] are architecturally mapped to AArch64 System register TRCACATR<n>[63:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR4.NUMACPAIRS) * 2 > n. Otherwise, direct accesses to TRCACATR<n> are RES0.

Attributes

TRCACATR<n> is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45
RES0													EXLEVEL_RL_EL2	EXLEVEL_RL_EL1	EXLEVEL_RL_EL0	RES0	EXLEVEL_NS_EL2	EXLEVEL_NS_EL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13

Bits [63:19]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [18]

When FEAT_RME is implemented:

Realm EL2 address comparison control. Controls whether a comparison can occur at EL2 in Realm state.

EXLEVEL_RL_EL2	Meaning
0b0	When TRCACATR<n>.EXLEVEL_NS_EL2 is 0 the Address Comparator performs comparisons in Realm EL2. When TRCACATR<n>.EXLEVEL_NS_EL2 is 1 the Address Comparator does not perform comparisons in Realm EL2.
0b1	When TRCACATR<n>.EXLEVEL_NS_EL2 is 0 the Address Comparator does not perform comparisons in Realm EL2. When TRCACATR<n>.EXLEVEL_NS_EL2 is 1 the Address Comparator performs comparisons in Realm EL2.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL1, bit [17]

When FEAT_RME is implemented:

Realm EL1 address comparison control. Controls whether a comparison can occur at EL1 in Realm state.

EXLEVEL_RL_EL1	Meaning
0b0	When TRCACATR<n>.EXLEVEL_NS_EL1 is 0 the Address Comparator performs comparisons in Realm EL1. When TRCACATR<n>.EXLEVEL_NS_EL1 is 1 the Address Comparator does not perform comparisons in Realm EL1.
0b1	When TRCACATR<n>.EXLEVEL_NS_EL1 is 0 the Address Comparator does not perform comparisons in Realm EL1. When TRCACATR<n>.EXLEVEL_NS_EL1 is 1 the Address Comparator performs comparisons in Realm EL1.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL0, bit [16]

When FEAT_RME is implemented:

Realm EL0 address comparison control. Controls whether a comparison can occur at EL0 in Realm state.

EXLEVEL_RL_EL0	Meaning
0b0	When TRCACATR<n>.EXLEVEL_NS_EL0 is 0 the Address Comparator performs comparisons in Realm EL0. When TRCACATR<n>.EXLEVEL_NS_EL0 is 1 the Address Comparator does not perform comparisons in Realm EL0.
0b1	When TRCACATR<n>.EXLEVEL_NS_EL0 is 0 the Address Comparator does not perform comparisons in Realm EL0. When TRCACATR<n>.EXLEVEL_NS_EL0 is 1 the Address Comparator performs comparisons in Realm EL0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [15]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [14]

When Non-secure EL2 is implemented:

Non-secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Non-secure state.

EXLEVEL_NS_EL2	Meaning
0b0	The Address Comparator performs comparisons in Non-secure EL2.
0b1	The Address Comparator does not perform comparisons in Non-secure EL2.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL1, bit [13]

When Non-secure EL1 is implemented:

Non-secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Non-secure state.

EXLEVEL_NS_EL1	Meaning
0b0	The Address Comparator performs comparisons in Non-secure EL1.
0b1	The Address Comparator does not perform comparisons in Non-secure EL1.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL0, bit [12]

When Non-secure EL0 is implemented:

Non-secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Non-secure state.

EXLEVEL_NS_EL0	Meaning
0b0	The Address Comparator performs comparisons in Non-secure EL0.
0b1	The Address Comparator does not perform comparisons in Non-secure EL0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [11]

When EL3 is implemented:

EL3 address comparison control. Controls whether a comparison can occur at EL3.

EXLEVEL_S_EL3	Meaning
0b0	The Address Comparator performs comparisons at EL3.
0b1	The Address Comparator does not perform comparisons at EL3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [10]
When Secure EL2 is implemented:

Secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Secure state.

EXLEVEL_S_EL2	Meaning
0b0	The Address Comparator performs comparisons in Secure EL2.
0b1	The Address Comparator does not perform comparisons in Secure EL2.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL1, bit [9]
When Secure EL1 is implemented:

Secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Secure state.

EXLEVEL_S_EL1	Meaning
0b0	The Address Comparator performs comparisons in Secure EL1.
0b1	The Address Comparator does not perform comparisons in Secure EL1.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [8]
When Secure EL0 is implemented:

Secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Secure state.

EXLEVEL_S_EL0	Meaning
0b0	The Address Comparator performs comparisons in Secure EL0.
0b1	The Address Comparator does not perform comparisons in Secure EL0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

CONTEXT, bits [6:4]

When TRCIDR4.NUMCIDC != 0b0000 or TRCIDR4.NUMVMIDC != 0b0000:

Selects a Context Identifier Comparator or Virtual Context Identifier Comparator:

CONTEXT	Meaning	Applies when
0b000	Comparator 0.	
0b001	Comparator 1.	When UInt(TRCIDR4.NUMCIDC) > 1 or UInt(TRCIDR4.NUMVMIDC) > 1
0b010	Comparator 2.	When UInt(TRCIDR4.NUMCIDC) > 2 or UInt(TRCIDR4.NUMVMIDC) > 2
0b011	Comparator 3.	When UInt(TRCIDR4.NUMCIDC) > 3 or UInt(TRCIDR4.NUMVMIDC) > 3
0b100	Comparator 4.	When UInt(TRCIDR4.NUMCIDC) > 4 or UInt(TRCIDR4.NUMVMIDC) > 4
0b101	Comparator 5.	When UInt(TRCIDR4.NUMCIDC) > 5 or UInt(TRCIDR4.NUMVMIDC) > 5
0b110	Comparator 6.	When UInt(TRCIDR4.NUMCIDC) > 6 or UInt(TRCIDR4.NUMVMIDC) > 6
0b111	Comparator 7.	When UInt(TRCIDR4.NUMCIDC) > 7 or UInt(TRCIDR4.NUMVMIDC) > 7

The width of this field is dependent on the maximum number of Context Identifier Comparators or Virtual Context Identifier Comparators implemented. Unimplemented bits are RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CONTEXTTYPE, bits [3:2]

When TRCIDR4.NUMCIDC != 0b0000 or TRCIDR4.NUMVMIDC != 0b0000:

Controls whether the Address Comparator is dependent on a Context Identifier Comparator, a Virtual Context Identifier Comparator, or both comparisons.

CONTEXTTYPE	Meaning	Applies when
0b00	The Address Comparator is not dependent on the Context Identifier Comparators or Virtual Context Identifier Comparators.	
0b01	The Address Comparator is dependent on the Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if both the Context Identifier Comparator and the address comparison match.	When TRCIDR4.NUMCIDC != 0b0000
0b10	The Address Comparator is dependent on the Virtual Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if both the Virtual Context Identifier Comparator and the address comparison match.	When TRCIDR4.NUMVMIDC != 0b0000
0b11	The Address Comparator is dependent on the Context Identifier Comparator and Virtual Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if the Context Identifier Comparator, the Virtual Context Identifier Comparator, and address comparison all match.	When TRCIDR4.NUMCIDC != 0b0000 and TRCIDR4.NUMVMIDC != 0b0000

If TRCIDR4.NUMCIDC == 0b0000, then bit [2] is RES0.

If TRCIDR4.NUMVMIDC == 0b0000, then bit [3] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [1:0]

Reserved, RES0.

Accessing TRCACATR<n>

Must be programmed if any of the following are true:

- TRCBBCTLR.RANGE[n/2] == 1.
- TRCRSCTLR<a>.GROUP == 0b0100 and TRCRSCTLR<a>.SAC[n] == 1.
- TRCRSCTLR<a>.GROUP == 0b0101 and TRCRSCTLR<a>.ARC[n/2] == 1.
- TRCVIIECTLR.EXCLUDE[n/2] == 1.
- TRCVIIECTLR.INCLUDE[n/2] == 1.
- TRCVISSCTLR.START[n] == 1.
- TRCVISSCTLR.STOP[n] == 1.
- TRCSSCCR<>.ARC[n/2] == 1.
- TRCSSCCR<>.SAC[n] == 1.
- TRCQCTLR.RANGE[n/2] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCACATR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0x480` + (8 * n)	TRCACATR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCACVR<n>, Trace Address Comparator Value Register <n>, n = 0 - 15

The TRCACVR<n> characteristics are:

Purpose

Contains the address value.

Configuration

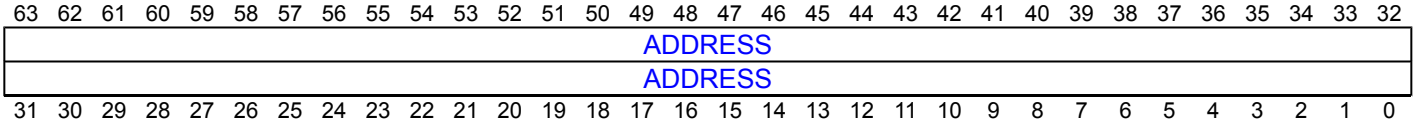
External register TRCACVR<n> bits [63:0] are architecturally mapped to AArch64 System register TRCACVR<n>[63:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR4.NUMACPAIRS) * 2 > n. Otherwise, direct accesses to TRCACVR<n> are RES0.

Attributes

TRCACVR<n> is a 64-bit register.

Field descriptions



ADDRESS, bits [63:0]

Address Value.

The Address Comparators can support implementations that use multiple address widths. When the trace unit compares the ADDRESS field with an address that has a width less than this field, then the address must be zero-extended to the ADDRESS field width. The trace unit then compares all implemented bits. For example, in a system that supports both 32-bit and 64-bit addresses, when the PE is in AArch32 state the comparator must zero-extend the 32-bit address and compare against the full 64 bits that are stored in TRCACVR<n>.ADDRESS. This requires that the trace analyzer always programs all implemented bits of TRCACVR<n>.ADDRESS.

The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an UNKNOWN value, where P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The result of writing a value of all zeros or all ones to ADDRESS at bits[63:P] is the written value, and a read of the register returns the written value.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCACVR<n>

Must be programmed if any of the following are true:

- TRCBBCTLR.RANGE[n/2] == 1.
- TRCRSCTLR<a>.GROUP == 0b0100 and TRCRSCTLR<a>.SAC[n] == 1.
- TRCRSCTLR<a>.GROUP == 0b0101 and TRCRSCTLR<a>.ARC[n/2] == 1.
- TRCVIIECTLR.EXCLUDE[n/2] == 1.

- TRCVIIECTLR.INCLUDE[n/2] == 1.
- TRCVISSCTLR.START[n] == 1.
- TRCVISSCTLR.STOP[n] == 1.
- TRCSSCCR<a>.ARC[n/2] == 1.
- TRCSSCCR<a>.SAC[n] == 1.
- TRCQCTLR.RANGE[n/2] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCACVR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0x400` + (8 * n)	TRCACVR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCAUTHSTATUS, Trace Authentication Status Register

The TRCAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

For additional information, see the CoreSight Architecture Specification.

Configuration

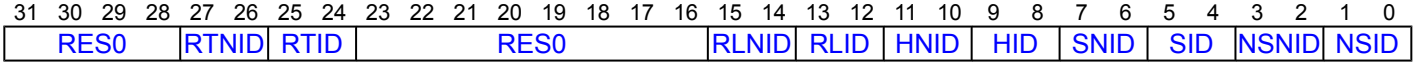
External register TRCAUTHSTATUS bits [31:0] are architecturally mapped to AArch64 System register TRCAUTHSTATUS[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCAUTHSTATUS are RES0.

Attributes

TRCAUTHSTATUS is a 32-bit register.

Field descriptions



Bits [31:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTNID.

RTID, bits [25:24]

Root invasive debug.

RTID	Meaning
0b00	Not implemented.

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLNID.

RLID, bits [13:12]

Realm invasive debug.

RLID	Meaning
0b00	Not implemented.

HNID, bits [11:10]

Hyp Non-invasive Debug. Indicates whether a separate enable control for EL2 non-invasive debug features is implemented and enabled.

HNID	Meaning
0b00	Separate Hyp non-invasive debug enable not implemented, or EL2 non-invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

HID, bits [9:8]

Hyp Invasive Debug. Indicates whether a separate enable control for EL2 invasive debug features is implemented and enabled.

HID	Meaning
0b00	Separate Hyp invasive debug enable not implemented, or EL2 invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

SNID, bits [7:6]

Secure Non-invasive Debug. Indicates whether Secure non-invasive debug features are implemented and enabled.

SNID	Meaning
0b00	Secure non-invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

When Secure state is implemented, this field reads as 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

When Secure state is not implemented, this field reads as 0b00.

SID, bits [5:4]

Secure Invasive Debug. Indicates whether Secure invasive debug features are implemented and enabled.

SID	Meaning
0b00	Secure invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

NSNID, bits [3:2]

Non-secure Non-invasive Debug. Indicates whether Non-secure non-invasive debug features are implemented and enabled.

NSNID	Meaning
0b00	Non-secure non-invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

When Non-secure state is implemented, this field reads as 0b11.

When Non-secure state is not implemented, this field reads as 0b00.

NSID, bits [1:0]

Non-secure Invasive Debug. Indicates whether Non-secure invasive debug features are implemented and enabled.

NSID	Meaning
0b00	Non-secure invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

Accessing TRCAUTHSTATUS

For implementations that support multiple access mechanisms, different access mechanisms can return different values for reads of TRCAUTHSTATUS if the authentication signals have changed and that change has not yet been synchronized by a Context synchronization event. This scenario can happen if, for example, the external debugger view is implemented separately from the system instruction view to allow for separate power domains, and so observes changes on the signals differently.

External debugger accesses to this register are unaffected by the OS Lock.

TRCAUTHSTATUS can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFB8	TRCAUTHSTATUS

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCAUXCTLR, Trace Auxiliary Control Register

The TRCAUXCTLR characteristics are:

Purpose

The function of this register is IMPLEMENTATION DEFINED.

Configuration

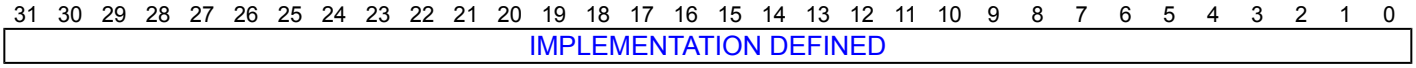
External register TRCAUXCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCAUXCTLR[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCAUXCTLR are RES0.

Attributes

TRCAUXCTLR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to `'00000000000000000000000000000000'`.

Accessing TRCAUXCTLR

If this register is nonzero then it might cause the behavior of a trace unit to contradict this architecture specification. See the documentation of the specific implementation for information about the IMPLEMENTATION DEFINED support for this register.

TRCAUXCTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x018	TRCAUXCTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCBBCTLR, Trace Branch Broadcast Control Register

The TRCBBCTLR characteristics are:

Purpose

Controls the regions in the memory map where branch broadcasting is active.

Configuration

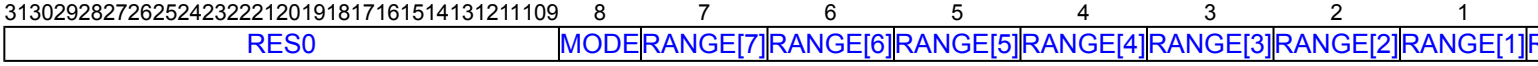
External register TRCBBCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCBBCTLR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, TRCIDR0.TRCBB == 1, and UInt(TRCIDR4.NUMACPAIRS) > 0. Otherwise, direct accesses to TRCBBCTLR are RES0.

Attributes

TRCBBCTLR is a 32-bit register.

Field descriptions



Bits [31:9]

Reserved, RES0.

MODE, bit [8]

Mode.

MODE	Meaning
0b0	Exclude Mode. Branch broadcasting is not active for instructions in the address ranges defined by TRCBBCTLR.RANGE. If TRCBBCTLR.RANGE == 0x00 then branch broadcasting is active for all instructions.
0b1	Include Mode. Branch broadcasting is active for instructions in the address ranges defined by TRCBBCTLR.RANGE. If TRCBBCTLR.RANGE == 0x00 then the behavior of the trace unit is CONSTRAINED UNPREDICTABLE. That is, the trace unit might or might not consider any instructions to be in a branch broadcasting region.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE[<m>], bit [m], for m = 7 to 0

Selects whether Address Range Comparator <m> is used with branch broadcasting.

RANGE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines, is not selected.
0b1	The address range that Address Range Comparator <m> defines, is selected.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCBBCTLR

Must be programmed if TRCCONFIGR.BB == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCBBCTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x03C	TRCBBCTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCCCTLR, Trace Cycle Count Control Register

The TRCCCCTLR characteristics are:

Purpose

Set the threshold value for cycle counting.

Configuration

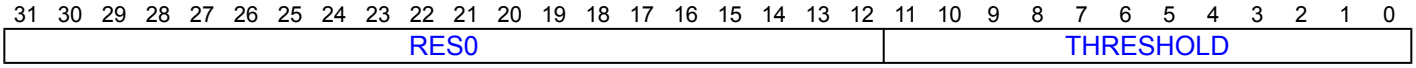
External register TRCCCCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCCCCTLR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and TRCIDR0.TRCCCI == 1. Otherwise, direct accesses to TRCCCCTLR are RES0.

Attributes

TRCCCCTLR is a 32-bit register.

Field descriptions



Bits [31:12]

Reserved, RES0.

THRESHOLD, bits [11:0]

Sets the threshold value for instruction trace cycle counting.

The minimum threshold value that can be programmed into THRESHOLD is given in TRCIDR3.CCITMIN. If the THRESHOLD value is smaller than the value in TRCIDR3.CCITMIN then the behavior is CONSTRAINED UNPREDICTABLE. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

Writing a value of zero when TRCCONFIGR.CCI enables instruction trace cycle counting results in CONSTRAINED UNPREDICTABLE behavior. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCCCTLR

Must be programmed if TRCCONFIGR.CCI == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCCCTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x038	TRCCCCTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDCCTLR0, Trace Context Identifier Comparator Control Register 0

The TRCCIDCCTLR0 characteristics are:

Purpose

Contains Context identifier mask values for the TRCCIDCVR<n> registers, for n = 0 to 3.

Configuration

External register TRCCIDCCTLR0 bits [31:0] are architecturally mapped to AArch64 System register TRCCIDCCTLR0[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, UInt(TRCIDR4.NUMCIDC) > 0x0, and UInt(TRCIDR2.CIDSIZE) > 0. Otherwise, direct accesses to TRCCIDCCTLR0 are RES0.

Attributes

TRCCIDCCTLR0 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20
COMP3[7]	COMP3[6]	COMP3[5]	COMP3[4]	COMP3[3]	COMP3[2]	COMP3[1]	COMP3[0]	COMP2[7]	COMP2[6]	COMP2[5]	COMP2[4]	C

COMP3[<m>], bit [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 3:

TRCCIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR3. Each bit in this field corresponds to a byte in TRCCIDCVR3.

COMP3[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR3[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR3[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP2[<m>], bit [m+16], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 2:

TRCCIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR2. Each bit in this field corresponds to a byte in TRCCIDCVR2.

COMP2[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP1[<m>], bit [m+8], for m = 7 to 0
When UInt(TRCIDR4.NUMCIDC) > 1:

TRCCIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR1. Each bit in this field corresponds to a byte in TRCCIDCVR1.

COMP1[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP0[<m>], bit [m], for m = 7 to 0
When UInt(TRCIDR4.NUMCIDC) > 0:

TRCCIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR0. Each bit in this field corresponds to a byte in TRCCIDCVR0.

COMP0[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.

- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Accessing TRCCIDCCTLR0

If software uses the TRCCIDCVR<n> registers, for n = 0 to 3, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCCIDCVR<n> to `0x00`.

If any bit is 1 and the relevant byte in TRCCIDCVR<n> is not `0x00`, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCIDCCTLR0 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0x680`	TRCCIDCCTLR0

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDCCTLR1, Trace Context Identifier Comparator Control Register 1

The TRCCIDCCTLR1 characteristics are:

Purpose

Contains Context identifier mask values for the TRCCIDCVR<n> registers, for n = 4 to 7.

Configuration

External register TRCCIDCCTLR1 bits [31:0] are architecturally mapped to AArch64 System register TRCCIDCCTLR1[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, UInt(TRCIDR4.NUMCIDC) > 0x4, and UInt(TRCIDR2.CIDSIZE) > 0. Otherwise, direct accesses to TRCCIDCCTLR1 are RES0.

Attributes

TRCCIDCCTLR1 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20
COMP7[7]	COMP7[6]	COMP7[5]	COMP7[4]	COMP7[3]	COMP7[2]	COMP7[1]	COMP7[0]	COMP6[7]	COMP6[6]	COMP6[5]	COMP6[4]	C

COMP7[<m>], bit [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 7:

TRCCIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR7. Each bit in this field corresponds to a byte in TRCCIDCVR7.

COMP7[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP6[<m>], bit [m+16], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 6:

TRCCIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR6. Each bit in this field corresponds to a byte in TRCCIDCVR6.

COMP6[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP5[<m>], bit [m+8], for m = 7 to 0
When UInt(TRCIDR4.NUMCIDC) > 5:

TRCCIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR5. Each bit in this field corresponds to a byte in TRCCIDCVR5.

COMP5[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP4[<m>], bit [m], for m = 7 to 0
When UInt(TRCIDR4.NUMCIDC) > 4:

TRCCIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR4. Each bit in this field corresponds to a byte in TRCCIDCVR4.

COMP4[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is **RES0**.

- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Accessing TRCCIDCCTLR1

If software uses the TRCCIDCVR<n> registers, for n = 4 to 7, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCCIDCVR<n> to `0x00`.

If any bit is 1 and the relevant byte in TRCCIDCVR<n> is not `0x00`, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCIDCCTLR1 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0x684`	TRCCIDCCTLR1

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDCVR<n>, Trace Context Identifier Comparator Value Registers <n>, n = 0 - 7

The TRCCIDCVR<n> characteristics are:

Purpose

Contains a Context identifier value.

Configuration

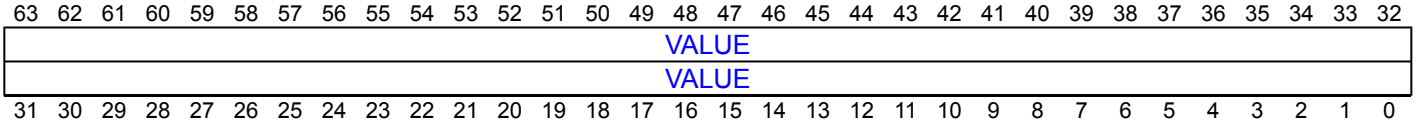
External register TRCCIDCVR<n> bits [63:0] are architecturally mapped to AArch64 System register TRCCIDCVR<n>[63:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR4.NUMCIDC) > n. Otherwise, direct accesses to TRCCIDCVR<n> are RES0.

Attributes

TRCCIDCVR<n> is a 64-bit register.

Field descriptions



VALUE, bits [63:0]

Context identifier value. The width of this field is indicated by TRCIDR2.CIDSIZE. Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Context identifier is zero until the PE updates the Context identifier.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCIDCVR<n>

Must be programmed if any of the following are true:

- TRCRSCTLR<a>.GROUP == 0b0110 and TRCRSCTLR<a>.CID[n] == 1.
- TRCACATR<a>.CONTEXTTYPE == 0b01 or 0b11 and TRCACATR<a>.CONTEXT == n.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCIDCVR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x600 + (8 * n)	TRCCIDCVR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDR0, Trace Component Identification Register 0

The TRCCIDR0 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

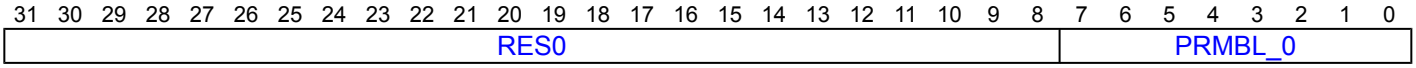
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCCIDR0 are RES0.

Attributes

TRCCIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Component identification preamble, segment 0.

Reads as 0x0D.

Access to this field is **RO**.

Accessing TRCCIDR0

External debugger accesses to this register are unaffected by the OS Lock.

TRCCIDR0 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFF0	TRCCIDR0

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDR1, Trace Component Identification Register 1

The TRCCIDR1 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

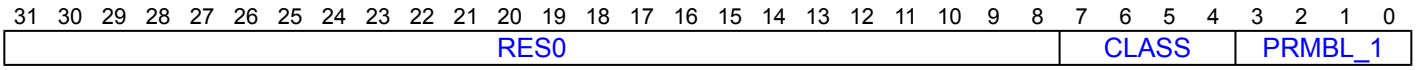
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCCIDR1 are RES0.

Attributes

TRCCIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

CLASS	Meaning
0b1001	CoreSight peripheral.

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

Access to this field is **RO**.

PRMBL_1, bits [3:0]

Component identification preamble, segment 1.

Reads as 0b0000.

Access to this field is **RO**.

Accessing TRCCIDR1

External debugger accesses to this register are unaffected by the OS Lock.

TRCCIDR1 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFF4	TRCCIDR1

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDR2, Trace Component Identification Register 2

The TRCCIDR2 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

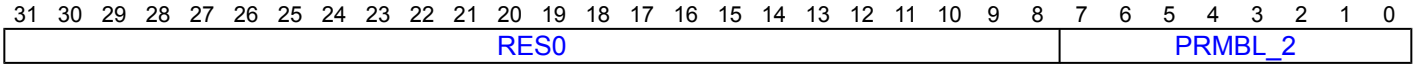
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCCIDR2 are RES0.

Attributes

TRCCIDR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Component identification preamble, segment 2.

Reads as `0x05`.

Access to this field is **RO**.

Accessing TRCCIDR2

External debugger accesses to this register are unaffected by the OS Lock.

TRCCIDR2 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0xFF8`	TRCCIDR2

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDR3, Trace Component Identification Register 3

The TRCCIDR3 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

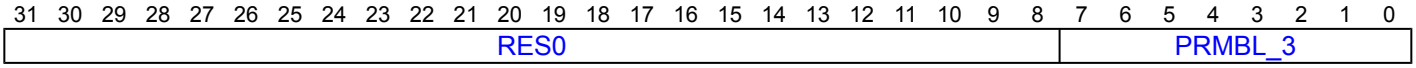
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCCIDR3 are RES0.

Attributes

TRCCIDR3 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Component identification preamble, segment 3.

Reads as `0xB1`.

Access to this field is **RO**.

Accessing TRCCIDR3

External debugger accesses to this register are unaffected by the OS Lock.

TRCCIDR3 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0xFFC`	TRCCIDR3

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCLAIMCLR, Trace Claim Tag Clear Register

The TRCCLAIMCLR characteristics are:

Purpose

In conjunction with TRCCLAIMSET, provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information, see the CoreSight Architecture Specification.

Configuration

External register TRCCLAIMCLR bits [31:0] are architecturally mapped to AArch64 System register TRCCLAIMCLR[31:0].

External register TRCCLAIMCLR bits [31:0] are architecturally mapped to AArch64 System register TRCCLAIMSET[31:0].

External register TRCCLAIMCLR bits [31:0] are architecturally mapped to External register TRCCLAIMSET[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCCLAIMCLR are RES0.

Attributes

TRCCLAIMCLR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	
CLR[31]	CLR[30]	CLR[29]	CLR[28]	CLR[27]	CLR[26]	CLR[25]	CLR[24]	CLR[23]	CLR[22]	CLR[21]	CLR[20]	CLR[19]	CLR[18]	CLR[17]	CL

CLR[<m>], bit [m], for m = 31 to 0

Claim Tag Clear. Indicates the current status of Claim Tag bit <m>, and is used to clear Claim Tag bit <m> to 0.

CLR[<m>]	Meaning
0b0	On a read: Claim Tag bit <m> is not set. On a write: Ignored.
0b1	On a read: Claim Tag bit <m> is set. On a write: Clear Claim tag bit <m> to 0.

The number of Claim Tag bits implemented is indicated in TRCCLAIMSET.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0'.

Accessing this field has the following behavior:

- When m >= NUM_TRC_CLAIM_TAGS, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing TRCCLAIMCLR

TRCCLAIMCLR can be accessed through the external debug interface:

Component	Offset	Instance

ETE	0xFA4	TRCCLAIMCLR

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCLAIMSET, Trace Claim Tag Set Register

The TRCCLAIMSET characteristics are:

Purpose

In conjunction with TRCCLAIMCLR, provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information, see the CoreSight Architecture Specification.

Configuration

External register TRCCLAIMSET bits [31:0] are architecturally mapped to AArch64 System register TRCCLAIMSET[31:0].

External register TRCCLAIMSET bits [31:0] are architecturally mapped to AArch64 System register TRCCLAIMCLR[31:0].

External register TRCCLAIMSET bits [31:0] are architecturally mapped to External register TRCCLAIMCLR[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCCLAIMSET are RES0.

The number of claim tag bits implemented is IMPLEMENTATION DEFINED. Arm recommends that implementations support a minimum of four claim tag bits, that is, SET[3:0] reads as `0b1111`.

Attributes

TRCCLAIMSET is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SET[31]	SET[30]	SET[29]	SET[28]	SET[27]	SET[26]	SET[25]	SET[24]	SET[23]	SET[22]	SET[21]	SET[20]	SET[19]	SET[18]	SET[17]	SET[

SET[<m>], bit [m], for m = 31 to 0

Claim Tag Set. Indicates whether Claim Tag bit <m> is implemented, and is used to set Claim Tag bit <m> to 1.

SET[<m>]	Meaning
`0b0`	On a read: Claim Tag bit <m> is not implemented. On a write: Ignored.
`0b1`	On a read: Claim Tag bit <m> is implemented. On a write: Set Claim Tag bit <m> to 1.

Accessing this field has the following behavior:

- When m >= NUM_TRC_CLAIM_TAGS, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RAO/W1S**.

Accessing TRCCLAIMSET

TRCCLAIMSET can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0xFA0`	TRCCLAIMSET

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCNTCTLR<n>, Trace Counter Control Register <n>, n = 0 - 3

The TRCCNTCTLR<n> characteristics are:

Purpose

Controls the operation of Counter <n>.

Configuration

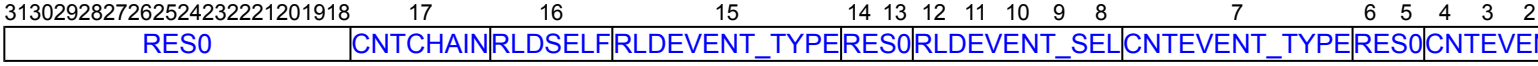
External register TRCCNTCTLR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCCNTCTLR<n>[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR5.NUMCNTR) > n. Otherwise, direct accesses to TRCCNTCTLR<n> are RES0.

Attributes

TRCCNTCTLR<n> is a 32-bit register.

Field descriptions



Bits [31:18]

Reserved, RES0.

CNTCHAIN, bit [17]

When n is odd:

For TRCCNTCTLR3 and TRCCNTCTLR1, this field controls whether the Counter decrements when a reload event occurs for Counter <n-1>.

CNTCHAIN	Meaning
0b0	The Counter does not decrement when a reload event for Counter <n-1> occurs.
0b1	Counter <n> decrements when a reload event for Counter <n-1> occurs. This concatenates Counter <n> and Counter <n-1>, to provide a larger count value.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLDSELF, bit [16]

Controls whether a reload event occurs for the Counter, when the Counter reaches zero.

RLDSELF	Meaning
0b0	Normal mode. The Counter is in Normal mode.
0b1	Self-reload mode. The Counter is in Self-reload mode.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RLDEVENT_TYPE, bit [15]

Selects an event, that when it occurs causes a reload event for Counter <n>

Chooses the type of Resource Selector.

RLDEVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCCNTCTLR<n>.RLDEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCCNTCTLR<n>.RLDEVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTLR<n>.RLDEVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

RLDEVENT_SEL, bits [12:8]

Selects an event, that when it occurs causes a reload event for Counter <n>

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR<n>.RLDEVENT.TYPE controls whether TRCCNTCTLR<n>.RLDEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

CNTEVENT_TYPE, bit [7]

Selects an event, that when it occurs causes Counter <n> to decrement.

Chooses the type of Resource Selector.

CNTEVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCCNTCTLR<n>.CNTEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCCNTCTLR<n>.CNTEVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTLR<n>.CNTEVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

CNTEVENT_SEL, bits [4:0]

Selects an event, that when it occurs causes Counter <n> to decrement.

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR<n>.CNTEVENT.TYPE controls whether TRCCNTCTLR<n>.CNTEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTCTLR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCNTCTLR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x150 + (4 * n)	TRCCNTCTLR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCNTRLDVR<n>, Trace Counter Reload Value Register <n>, n = 0 - 3

The TRCCNTRLDVR<n> characteristics are:

Purpose

This sets or returns the reload count value for Counter <n>.

Configuration

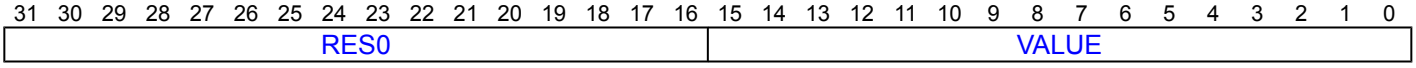
External register TRCCNTRLDVR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCCNTRLDVR<n>[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR5.NUMCNTR) > n. Otherwise, direct accesses to TRCCNTRLDVR<n> are RES0.

Attributes

TRCCNTRLDVR<n> is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

VALUE, bits [15:0]

Contains the reload value for Counter <n>. When a reload event occurs for Counter <n> then the trace unit copies the VALUE<n> field into Counter <n>.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTRLDVR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCNTRLDVR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x140 + (4 * n)	TRCCNTRLDVR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCNTVR<n>, Trace Counter Value Register <n>, n = 0 - 3

The TRCCNTVR<n> characteristics are:

Purpose

This sets or returns the value of Counter <n>.

Configuration

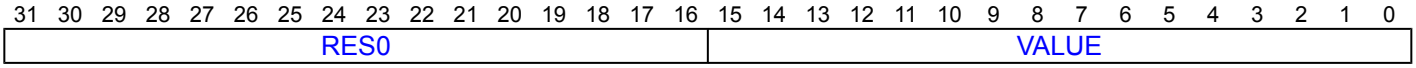
External register TRCCNTVR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCCNTVR<n>[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR5.NUMCNTR) > n. Otherwise, direct accesses to TRCCNTVR<n> are RES0.

Attributes

TRCCNTVR<n> is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

VALUE, bits [15:0]

Contains the count value of Counter.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTVR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCCNTVR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x160 + (4 * n)	TRCCNTVR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCONFIGR, Trace Configuration Register

The TRCCONFIGR characteristics are:

Purpose

Controls the tracing options.

Configuration

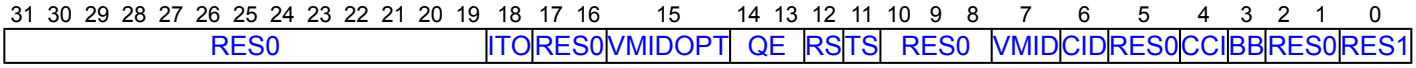
External register TRCCONFIGR bits [31:0] are architecturally mapped to AArch64 System register TRCCONFIGR[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCCONFIGR are RES0.

Attributes

TRCCONFIGR is a 32-bit register.

Field descriptions



Bits [31:19]

Reserved, RES0.

ITO, bit [18]

When TRCIDR0.ITE == 1:

Instrumentation Trace Override.

ITO	Meaning
0b0	Instrumentation Trace Override disabled.
0b1	Instrumentation Trace Override enabled.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [17:16]

Reserved, RES0.

VMIDOPT, bit [15]

When TRCIDR2.VMIDOPT == 0b01:

Virtual context identifier selection control.

VMIDOPT	Meaning
0b0	VTTBR_EL2.VMID is used as the Virtual context identifier.
0b1	CONTEXTIDR_EL2.PROCID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == 0b00:

Reserved, RES0.

Virtual context identifier selection control.

VTTBR_EL2.VMID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == 0b10:

Reserved, RES1.

Virtual context identifier selection control.

CONTEXTIDR_EL2.PROCID is used as the Virtual context identifier.

Otherwise:

Reserved, RES0.

QE, bits [14:13]

When TRCIDR0.QSUPP == 0b01:

Q element generation control.

QE	Meaning
0b00	Q elements are disabled.
0b01	Q elements with instruction counts are enabled.
	Q elements without instruction counts are disabled.

All other values are reserved.

When TRCIDR0.QSUPP == 0b10:

Q element generation control.

QE	Meaning
0b00	Q elements are disabled.
0b11	Q elements with instruction counts are enabled.
	Q elements without instruction counts are enabled.

All other values are reserved.

When TRCIDR0.QSUPP == 0b11:

Q element generation control.

QE	Meaning
0b00	Q elements are disabled.
0b01	Q elements with instruction counts are enabled. Q elements without instruction counts are disabled.
0b11	Q elements with instruction counts are enabled. Q elements without instruction counts are enabled.

All other values are reserved.

Otherwise:

Reserved, RES0.

RS, bit [12]
When TRCIDR0.RETSTACK == 1:

Return stack control.

RS	Meaning
0b0	Return stack is disabled.
0b1	Return stack is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TS, bit [11]
When TRCIDR0.TSSIZE != 0b00000:

Global timestamp tracing control.

TS	Meaning
0b0	Global timestamp tracing is disabled.
0b1	Global timestamp tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [10:8]

Reserved, RES0.

VMID, bit [7]
When TRCIDR2.VMIDSIZE != 0b00000:

Virtual context identifier tracing control.

VMID	Meaning
0b0	Virtual context identifier tracing is disabled.
0b1	Virtual context identifier tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CID, bit [6]
When TRCIDR2.CIDSIZE != 0b00000:

Context identifier tracing control.

CID	Meaning
0b0	Context identifier tracing is disabled.
0b1	Context identifier tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [5]

Reserved, RES0.

CCI, bit [4]
When TRCIDR0.TRCCCI == 1:

Cycle counting instruction tracing control.

CCI	Meaning
0b0	Cycle counting instruction tracing is disabled.
0b1	Cycle counting instruction tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BB, bit [3]
When TRCIDR0.TRCBB == 1:

Branch broadcasting control.

BB	Meaning
0b0	Branch broadcasting is disabled.
0b1	Branch broadcasting is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [2:1]

Reserved, RES0.

Bit [0]

Reserved, RES1.

Accessing TRCCONFIGR

Must always be programmed.

TRCCONFIGR.QE must be set to `0b00` if TRCCONFIGR.BB is not 0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCONFIGR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0x010`	TRCCONFIGR

Accessible as follows:

- When OSLockStatus(), or !IsTraceCorePowered(), or !AllowExternalTraceAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVAFF, Trace Device Affinity Register

The TRCDEVAFF characteristics are:

Purpose

For additional information, see the CoreSight Architecture Specification.

Reads the same value as the MPIDR_EL1 register for the PE that this trace unit has affinity with.

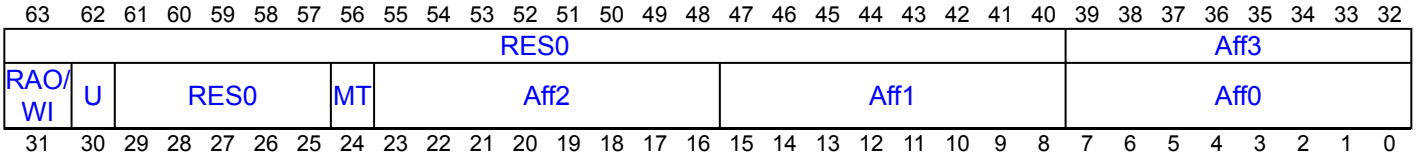
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCDEVAFF are RES0.

Attributes

TRCDEVAFF is a 64-bit register.

Field descriptions



Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [31]

Reserved, RAO/WI.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Access to this field is **RO**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is **RO**.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCDEVAFF

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVAFF can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFA8	TRCDEVAFF

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.

- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVARCH, Trace Device Architecture Register

The TRCDEVARCH characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

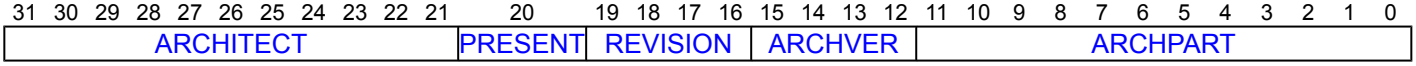
External register TRCDEVARCH bits [31:0] are architecturally mapped to AArch64 System register TRCDEVARCH[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCDEVARCH are RES0.

Attributes

TRCDEVARCH is a 32-bit register.

Field descriptions



ARCHITECT, bits [31:21]

Defines the architect of the component. For Trace, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, `0b0100`.

Bits [27:21] are the JEP106 identification code, `0b0111011`.

Reads as `0b01000111011`.

Access to this field is **RO**.

PRESENT, bit [20]

DEVARCH present. Indicates that the TRCDEVARCH register is present.

Reads as `0b1`.

Access to this field is **RO**.

REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

REVISION	Meaning
0b0000	ETEv1.0, FEAT_ETE.
0b0001	ETEv1.1, FEAT_ETEv1p1.
0b0010	ETEv1.2, FEAT_ETEv1p2.
0b0011	ETEv1.3, FEAT_ETEv1p3.

All other values are reserved.

Access to this field is **RO**.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

ARCHVER	Meaning
0b0101	ETEv1.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12].

This field reads as 0x5.

Access to this field is **RO**.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

ARCHPART	Meaning
0xA13	Arm PE trace architecture.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0].

This field reads as 0xA13.

Access to this field is **RO**.

Accessing TRCDEVARCH

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVARCH can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFBC	TRCDEVARCH

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVID, Trace Device Configuration Register

The TRCDEVID characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

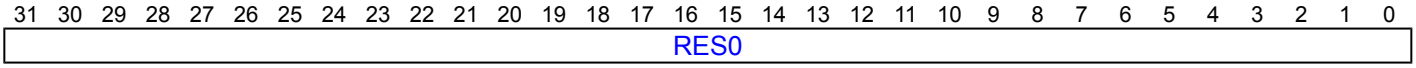
External register TRCDEVID bits [31:0] are architecturally mapped to AArch64 System register TRCDEVID[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCDEVID are RES0.

Attributes

TRCDEVID is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRCDEVID

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVID can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFC8	TRCDEVID

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVID1, Trace Device Configuration Register 1

The TRCDEVID1 characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

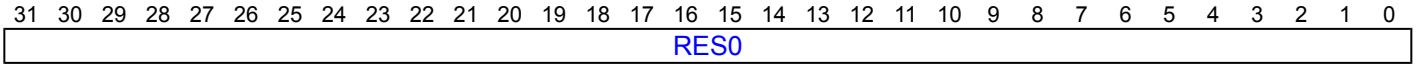
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCDEVID1 are RES0.

Attributes

TRCDEVID1 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRCDEVID1

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVID1 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFC4	TRCDEVID1

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVID2, Trace Device Configuration Register 2

The TRCDEVID2 characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

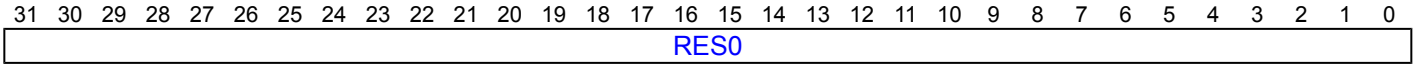
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCDEVID2 are RES0.

Attributes

TRCDEVID2 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRCDEVID2

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVID2 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFC0	TRCDEVID2

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVTYPE, Trace Device Type Register

The TRCDEVTYPE characteristics are:

Purpose

Provides discovery information for the component. If the part number field is not recognized, a debugger can report the information that is provided by TRCDEVTYPE about the component instead.

For additional information, see the CoreSight Architecture Specification.

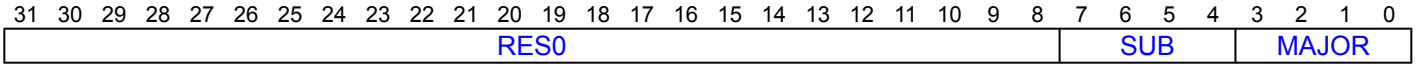
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCDEVTYPE are RES0.

Attributes

TRCDEVTYPE is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Component sub-type.

SUB	Meaning
0b0001	When MAJOR == 0x3 (Trace source): Associated with a PE.

This field reads as 0x1.

Access to this field is **RO**.

MAJOR, bits [3:0]

Component major type.

MAJOR	Meaning
0b0011	Trace source.

Other values are defined by the CoreSight Architecture.

This field reads as 0x3.

Access to this field is **RO**.

Accessing TRCDEVTYPE

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVTYPE can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFCC	TRCDEVTYPE

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEVENTCTL0R, Trace Event Control 0 Register

The TRCEVENTCTL0R characteristics are:

Purpose

Controls the generation of ETEEvents.

Configuration

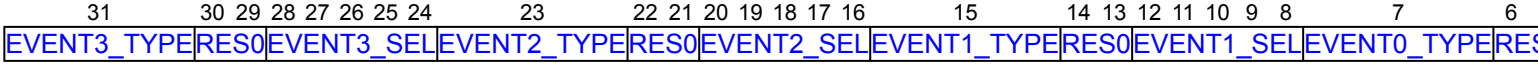
External register TRCEVENTCTL0R bits [31:0] are architecturally mapped to AArch64 System register TRCEVENTCTL0R[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and TRCIDR4.NUMRSPAIR != 0b0000. Otherwise, direct accesses to TRCEVENTCTL0R are RES0.

Attributes

TRCEVENTCTL0R is a 32-bit register.

Field descriptions



EVENT3_TYPE, bit [31]

When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 3:

Chooses the type of Resource Selector.

EVENT3_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT3.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT3.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [30:29]

Reserved, RES0.

EVENT3_SEL, bits [28:24]

When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 3:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[3] == 1, then Event element 3 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.TYPE controls whether TRCEVENTCTL0R.EVENT3.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT2_TYPE, bit [23]

When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 2:

Chooses the type of Resource Selector.

EVENT2_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT2.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT2.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [22:21]

Reserved, RES0.

EVENT2_SEL, bits [20:16]

When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 2:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[2] == 1, then Event element 2 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.TYPE controls whether TRCEVENTCTL0R.EVENT2.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT1_TYPE, bit [15]
When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 1:

Chooses the type of Resource Selector.

EVENT1_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT1.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT1.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [14:13]

Reserved, RES0.

EVENT1_SEL, bits [12:8]
When TRCIDR4.NUMRSPAIR != 0b0000 and UInt(TRCIDR0.NUMEVENT) >= 1:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[1] == 1, then Event element 1 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.TYPE controls whether TRCEVENTCTL0R.EVENT1.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT0_TYPE, bit [7]
When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

EVENT0_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT0.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT0.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT0_SEL, bits [4:0]
When TRCIDR4.NUMRSPAIR != 0b0000:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[0] == 1, then Event element 0 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.TYPE controls whether TRCEVENTCTL0R.EVENT0.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCEVENTCTL0R

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCEVENTCTL0R can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x020	TRCEVENTCTL0R

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEVENTCTL1R, Trace Event Control 1 Register

The TRCEVENTCTL1R characteristics are:

Purpose

Controls the behavior of the ETEEvents that TRCEVENTCTL0R selects.

Configuration

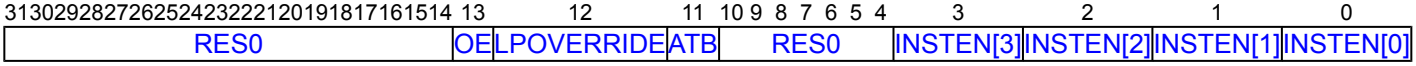
External register TRCEVENTCTL1R bits [31:0] are architecturally mapped to AArch64 System register TRCEVENTCTL1R[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCEVENTCTL1R are RES0.

Attributes

TRCEVENTCTL1R is a 32-bit register.

Field descriptions



Bits [31:14]

Reserved, RES0.

OE, bit [13]

When TRCIDR5.OE == 1:

ETE Trace Output Enable control.

OE	Meaning
0b0	Trace output to any IMPLEMENTATION DEFINED trace output interface is disabled.
0b1	Trace output to any IMPLEMENTATION DEFINED trace output interface is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

LPOVERRIDE, bit [12]

When TRCIDR5.LPOVERRIDE == 1:

Low-power Override Mode select.

LPOVERRIDE	Meaning
0b0	Trace unit Low-power Override Mode is not enabled. That is, the trace unit is permitted to enter low-power state.
0b1	Trace unit Low-power Override Mode is enabled. That is, entry to a low-power state does not affect the trace unit resources or trace generation.

Otherwise:

Reserved, RES0.

ATB, bit [11]

When TRCIDR5.ATBTRIG == 1:

AMBA Trace Bus (ATB) trigger enable.

If a CoreSight ATB interface is implemented then when ETEEvent 0 occurs the trace unit sets:

- ATID == 0x7D.
- ATDATA to the value of TRCTRACEIDR.

If the width of ATDATA is greater than the width of TRCTRACEIDR.TRACEID then the trace unit zeros the upper ATDATA bits.

If ETEEvent 0 is programmed to occur based on program execution, such as an Address Comparator, the ATB trigger might not be inserted into the ATB stream at the same time as any trace generated by that program execution is output by the trace unit. Typically, the generated trace might be buffered in a trace unit which means that the ATB trigger would be output before the associated trace is output.

If ETEEvent 0 is asserted multiple times in close succession, the trace unit is required to generate an ATB trigger for the first assertion, but might ignore one or more of the subsequent assertions. Arm recommends that the window in which ETEEvent 0 is ignored is limited only by the time taken to output an ATB trigger.

ATB	Meaning
0b0	ATB trigger is disabled.
0b1	ATB trigger is enabled.

Otherwise:

Reserved, RES0.

Bits [10:4]

Reserved, RES0.

INSTEN[<m>], bit [m], for m = 3 to 0

Event element control.

INSTEN[<m>]	Meaning
0b0	The trace unit does not generate an Event element <m>.
0b1	The trace unit generates an Event element <m> when ETEEvent <m> occurs.

Accessing this field has the following behavior:

- When TRCIDR4.NUMRSPAIR == 0b0000, access to this field is **RES0**.
- Access to this field is **RES0** if all the following are true:
  - TRCIDR4.NUMRSPAIR != 0b0000.
  - m > UInt(TRCIDR0.NUMEVENT).
- Otherwise, access to this field is **RW**.

Accessing TRCEVENTCTL1R

Must be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCEVENTCTL1R can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x024	TRCEVENTCTL1R

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEXTINSELR<n>, Trace External Input Select Register <n>, n = 0 - 3

The TRCEXTINSELR<n> characteristics are:

Purpose

Use this to set, or read, which External Inputs are resources to the trace unit.

The name TRCEXTINSELR is an alias of TRCEXTINSELR0.

Configuration

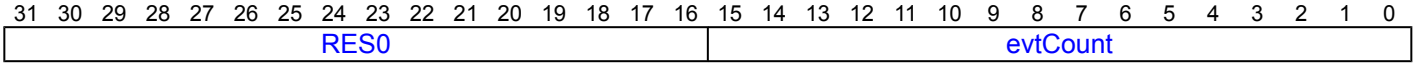
External register TRCEXTINSELR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCEXTINSELR<n>[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR5.NUMEXTINSEL) > n. Otherwise, direct accesses to TRCEXTINSELR<n> are RES0.

Attributes

TRCEXTINSELR<n> is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

evtCount, bits [15:0]

PMU event to select.

The event number as defined by the Arm ARM.

Software must program this field with a PMU event that is supported by the PE being programmed.

There are three ranges of PMU event numbers:

- PMU event numbers in the range `0x0000` to `0x003F` are common architectural and microarchitectural events.
- PMU event numbers in the range `0x0040` to `0x00BF` are Arm recommended common architectural and microarchitectural PMU events.
- PMU event numbers in the range `0x00C0` to `0x03FF` are IMPLEMENTATION DEFINED PMU events.

If evtCount is programmed to a PMU event that is reserved or not supported by the PE, the behavior depends on the PMU event type:

- For the range `0x0000` to `0x003F`, then the PMU event is not active, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For IMPLEMENTATION DEFINED PMU events, it is UNPREDICTABLE what PMU event, if any, is counted, and the value returned by a direct or external read of the evtCount field is UNKNOWN.

UNPREDICTABLE means the PMU event must not expose privileged information.

Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include a PMU event from a set of common IMPLEMENTATION DEFINED PMU events, then no PMU event is counted and the value read back on evtCount is the value written.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCEXTINSELR<n>

Must be programmed if any of the following is true: TRCRSCTLR<a>.GROUP == 0b0000 and TRCRSCTLR<a>.EXTIN[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCEXTINSELR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x120 + (4 * n)	TRCEXTINSELR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR0, Trace ID Register 0

The TRCIDR0 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

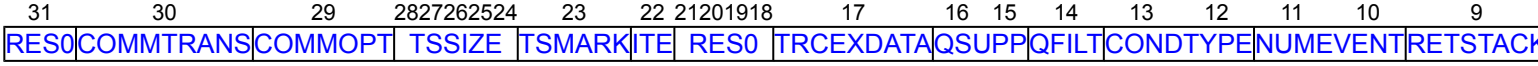
External register TRCIDR0 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR0[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR0 are RES0.

Attributes

TRCIDR0 is a 32-bit register.

Field descriptions



Bit [31]

Reserved, RES0.

COMMTRANS, bit [30]

Transaction Start element behavior.

The value of this field is an IMPLEMENTATION DEFINED choice of:

COMMTRANS	Meaning
0b0	Transaction Start elements are P0 elements.
0b1	Transaction Start elements are not P0 elements.

Access to this field is **RO**.

COMMOPT, bit [29]

Indicates the contents and encodings of Cycle count packets.

The value of this field is an IMPLEMENTATION DEFINED choice of:

COMMOPT	Meaning
0b0	Commit mode 0.
0b1	Commit mode 1.

The Commit mode defines the contents and encodings of Cycle Count packets, in particular how Commit elements are indicated by these packets. See the descriptions of these packets for more details.

Accessing this field has the following behavior:

- Access to this field is **RAO/WI** if all the following are true:
  - TRCIDR0.TRCCCI == 1.
  - UInt(TRCIDR8.MAXSPEC) == 0x0.

- When TRCIDR0.TRCCCI == 0, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RO**.

TSSIZE, bits [28:24]

Indicates that the trace unit implements Global timestamping and the size of the timestamp value.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TSSIZE	Meaning
0b00000	Global timestamping not implemented.
0b01000	Global timestamping implemented with a 64-bit timestamp value.

All other values are reserved.

This field reads as 0b01000.

Access to this field is **RO**.

TSMARK, bit [23]

Indicates whether Timestamp Marker elements are generated.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TSMARK	Meaning
0b0	Timestamp Marker elements are not generated.
0b1	Timestamp Marker elements are generated.

Access to this field is **RO**.

ITE, bit [22]

Indicates whether Instrumentation Trace is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ITE	Meaning
0b0	Instrumentation Trace not implemented.
0b1	Instrumentation Trace implemented.

This field has the value 1 if FEAT_ITE is implemented.

Access to this field is **RO**.

Bits [21:18]

Reserved, RES0.

TRCEXDATA, bit [17]

When TRCIDR0.TRCDATA != 0b00:

Indicates if the trace unit implements tracing of data transfers for exceptions and exception returns. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCEXDATA	Meaning
0b0	Tracing of data transfers for exceptions and exception returns not implemented.
0b1	Tracing of data transfers for exceptions and exception returns implemented.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

QSUPP, bits [16:15]

Indicates that the trace unit implements Q element support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

QSUPP	Meaning
0b00	Q element support is not implemented.
0b01	Q element support is implemented, and only supports Q elements with instruction counts.
0b10	Q element support is implemented, and only supports Q elements without instruction counts.
0b11	Q element support is implemented, and supports: • Q elements with instruction counts. • Q elements without instruction counts.

Access to this field is **RO**.

QFILT, bit [14]

Indicates if the trace unit implements Q element filtering.

The value of this field is an IMPLEMENTATION DEFINED choice of:

QFILT	Meaning
0b0	Q element filtering is not implemented.
0b1	Q element filtering is implemented.

If TRCIDR0.QSUPP == 0b00 then this field is 0.

Access to this field is **RO**.

CONDTYPE, bits [13:12]

When TRCIDR0.TRCCOND == 1:

Indicates how conditional instructions are traced. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CONDTYPE	Meaning
0b00	Conditional instructions are traced with an indication of whether they pass or fail their condition code check.
0b01	Conditional instructions are traced with an indication of the APSR condition flags.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

NUMEVENT, bits [11:10]

When TRCIDR4.NUMRSPAIR == 0b0000:

Indicates the number of ETEEvents implemented.

NUMEVENT	Meaning
0b00	The trace unit supports 0 ETEEvents.

All other values are reserved.

Access to this field is **RO**.

When TRCIDR4.NUMRSPAIR != 0b0000:

Indicates the number of ETEEvents implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMEVENT	Meaning
0b00	The trace unit supports 1 ETEEvent.
0b01	The trace unit supports 2 ETEEvents.
0b10	The trace unit supports 3 ETEEvents.
0b11	The trace unit supports 4 ETEEvents.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

RETSTACK, bit [9]

Indicates if the trace unit supports the return stack.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RETSTACK	Meaning
0b0	Return stack not implemented.
0b1	Return stack implemented.

Access to this field is **RO**.

Bit [8]

Reserved, RES0.

TRCCCI, bit [7]

Indicates if the trace unit implements cycle counting.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCCCI	Meaning
0b0	Cycle counting not implemented.
0b1	Cycle counting implemented.

This field reads as 1.

Access to this field is **RO**.

TRCCOND, bit [6]

Indicates if the trace unit implements conditional instruction tracing. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCCOND	Meaning
0b0	Conditional instruction tracing not implemented.
0b1	Conditional instruction tracing implemented.

This field reads as 0.

Access to this field is **RO**.

TRCBB, bit [5]

Indicates if the trace unit implements branch broadcasting.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCBB	Meaning
0b0	Branch broadcasting not implemented.
0b1	Branch broadcasting implemented.

This field reads as 1.

Access to this field is **RO**.

TRCDATA, bits [4:3]

Indicates if the trace unit implements data tracing. Data tracing is not implemented in ETE and this field is reserved for other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCDATA	Meaning
0b00	Data tracing not implemented.
0b11	Data tracing implemented.

All other values are reserved.

This field reads as 0b00.

Access to this field is **RO**.

INSTP0, bits [2:1]

Indicates if load and store instructions are P0 instructions. Load and store instructions as P0 instructions is not implemented in ETE and this field is reserved for other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

INSTP0	Meaning
0b00	Load and store instructions are not P0 instructions.
0b11	Load and store instructions are P0 instructions.

All other values are reserved.

When FEAT_ETE is implemented, the only permitted value is 0b00.

Access to this field is **RO**.

Bit [0]

Reserved, RES1.

Accessing TRCIDR0

TRCIDR0 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x1E0	TRCIDR0

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR1, Trace ID Register 1

The TRCIDR1 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

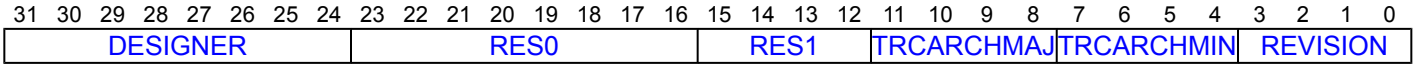
External register TRCIDR1 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR1[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR1 are RES0.

Attributes

TRCIDR1 is a 32-bit register.

Field descriptions



DESIGNER, bits [31:24]

Indicates which company designed the trace unit. The permitted values of this field are the same as MIDR_EL1.Implementer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:16]

Reserved, RES0.

Bits [15:12]

Reserved, RES1.

TRCARCHMAJ, bits [11:8]

Major architecture version.

TRCARCHMAJ	Meaning
`0b1111`	If both TRCIDR1.TRCARCHMAJ and TRCIDR1.TRCARCHMIN == `0xF` then refer to TRCDEVARCH.

All other values are reserved.

This field reads as `0b1111`.

Access to this field is **RO**.

TRCARCHMIN, bits [7:4]

Minor architecture version.

TRCARCHMIN	Meaning
0b1111	If both TRCIDR1.TRCARCHMAJ and TRCIDR1.TRCARCHMIN == 0xF then refer to TRCDEVARCH.

All other values are reserved.

This field reads as 0b1111.

Access to this field is **RO**.

REVISION, bits [3:0]

Implementation revision.

Returns an IMPLEMENTATION DEFINED value that identifies the revision of the trace unit.

Arm deprecates any use of this field and recommends that implementations set this field to zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCIDR1

TRCIDR1 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x1E4	TRCIDR1

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR10, Trace ID Register 10

The TRCIDR10 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

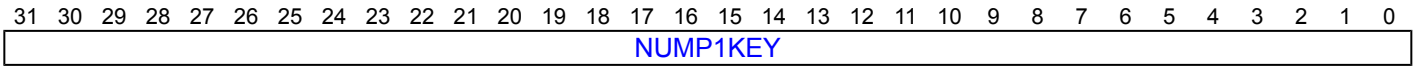
External register TRCIDR10 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR10[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR10 are RES0.

Attributes

TRCIDR10 is a 32-bit register.

Field descriptions



NUMP1KEY, bits [31:0]

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR10

TRCIDR10 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x188	TRCIDR10

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR11, Trace ID Register 11

The TRCIDR11 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

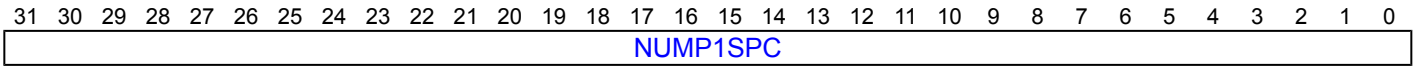
External register TRCIDR11 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR11[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR11 are RES0.

Attributes

TRCIDR11 is a 32-bit register.

Field descriptions



NUMP1SPC, bits [31:0]

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of special P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR11

TRCIDR11 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x18C	TRCIDR11

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR12, Trace ID Register 12

The TRCIDR12 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

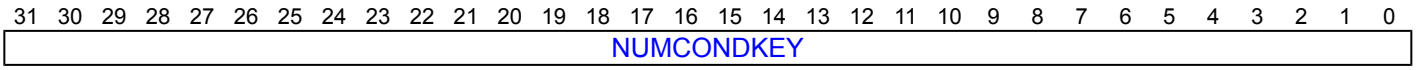
External register TRCIDR12 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR12[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR12 are RES0.

Attributes

TRCIDR12 is a 32-bit register.

Field descriptions



NUMCONDKEY, bits [31:0]

When TRCIDR0.TRCCOND == 1:

Indicates the number of conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR12

TRCIDR12 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x190	TRCIDR12

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR13, Trace ID Register 13

The TRCIDR13 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

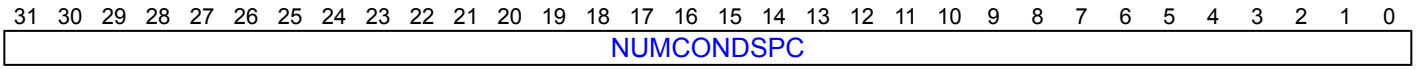
External register TRCIDR13 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR13[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR13 are RES0.

Attributes

TRCIDR13 is a 32-bit register.

Field descriptions



NUMCONDSPC, bits [31:0]

When TRCIDR0.TRCCOND == 1:

Indicates the number of special conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR13

TRCIDR13 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x194	TRCIDR13

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR2, Trace ID Register 2

The TRCIDR2 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

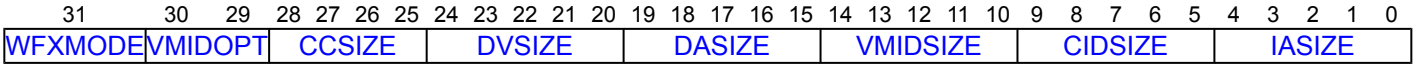
External register TRCIDR2 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR2[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR2 are RES0.

Attributes

TRCIDR2 is a 32-bit register.

Field descriptions



WFXMODE, bit [31]

Indicates whether `WFI`, `WFIT`, `WFE`, and `WFET` instructions are classified as P0 instructions:

The value of this field is an IMPLEMENTATION DEFINED choice of:

WFXMODE	Meaning
`0b0`	`WFI`, `WFIT`, `WFE`, and `WFET` instructions are not classified as P0 instructions.
`0b1`	`WFI`, `WFIT`, `WFE`, and `WFET` instructions are classified as P0 instructions.

Access to this field is **RO**.

VMIDOPT, bits [30:29]

Indicates the options for Virtual context identifier selection.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMIDOPT	Meaning
`0b00`	Virtual context identifier selection not supported. TRCCONFIGR.VMIDOPT is RES0.
`0b01`	Virtual context identifier selection supported. TRCCONFIGR.VMIDOPT is implemented.
`0b10`	Virtual context identifier selection not supported. TRCCONFIGR.VMIDOPT is RES1.

All other values are reserved.

If TRCIDR2.VMIDSIZE == `0b00000` then this field is `0b00`.

If TRCIDR2.VMIDSIZE != `0b00000` then this field is `0b10`.

Access to this field is **RO**.

CCSIZE, bits [28:25]

When TRCIDR0.TRCCCI == 1:

Indicates the size of the cycle counter.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCSIZE	Meaning
0b0000	The cycle counter is 12 bits in length.
0b0001	The cycle counter is 13 bits in length.
0b0010	The cycle counter is 14 bits in length.
0b0011	The cycle counter is 15 bits in length.
0b0100	The cycle counter is 16 bits in length.
0b0101	The cycle counter is 17 bits in length.
0b0110	The cycle counter is 18 bits in length.
0b0111	The cycle counter is 19 bits in length.
0b1000	The cycle counter is 20 bits in length.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

DVSIZE, bits [24:20]

When TRCIDR0.TRCDATA != 0b00:

Indicates the data value size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DVSIZE	Meaning
0b00000	Data value tracing not implemented.
0b00100	Data value tracing has a maximum of 32-bit data values.
0b01000	Data value tracing has a maximum of 64-bit data values.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

DASIZE, bits [19:15]

When TRCIDR0.TRCDATA != 0b00:

Indicates the data address size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DASIZE	Meaning
0b00000	Data address tracing not implemented.
0b00100	Data address tracing has a maximum of 32-bit data addresses.
0b01000	Data address tracing has a maximum of 64-bit data addresses.

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

VMIDSIZE, bits [14:10]

Indicates the trace unit Virtual context identifier size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMIDSIZE	Meaning
0b00000	Virtual context identifier tracing is not supported.
0b00001	8-bit Virtual context identifier size.
0b00010	16-bit Virtual context identifier size.
0b00100	32-bit Virtual context identifier size.

All other values are reserved.

If the PE does not implement EL2 then this field is 0b00000.

If the PE implements EL2 then this field is 0b00100.

Access to this field is **RO**.

CIDSIZE, bits [9:5]

Indicates the Context identifier size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CIDSIZE	Meaning
0b00000	Context identifier tracing is not supported.
0b00100	32-bit Context identifier size.

All other values are reserved.

This field reads as 0b00100.

Access to this field is **RO**.

IASIZE, bits [4:0]

Virtual instruction address size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IASIZE	Meaning
0b00100	Maximum of 32-bit instruction address size.
0b01000	Maximum of 64-bit instruction address size.

All other values are reserved.

This field reads as 0b01000.

Access to this field is **RO**.

Accessing TRCIDR2

TRCIDR2 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x1E8	TRCIDR2

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR3, Trace ID Register 3

The TRCIDR3 characteristics are:

Purpose

Returns the base architecture of the trace unit.

Configuration

External register TRCIDR3 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR3[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR3 are RES0.

Attributes

TRCIDR3 is a 32-bit register.

Field descriptions

NOOVERFLOW, bit [31]

Indicates if overflow prevention is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NOOVERFLOW	Meaning
0b0	Overflow prevention is not implemented.
0b1	Overflow prevention is implemented.

If TRCIDR3.STALLCTL == 0 then this field is 0.

Access to this field is **RO**.

NUMPROC, bits [13:12, 30:28]

Indicates the number of PEs available for tracing.

NUMPROC	Meaning
0b00000	The trace unit can trace one PE.

This field reads as 0b00000.

The NUMPROC field is split as follows:

- NUMPROC[2:0] is TRCIDR3[30:28].
- NUMPROC[4:3] is TRCIDR3[13:12].

Access to this field is **RO**.

SYSSTALL, bit [27]

Indicates if stalling of the PE is permitted.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYSSTALL	Meaning
0b0	Stalling of the PE is not permitted.
0b1	Stalling of the PE is permitted.

The value of this field might be dynamic and change based on system conditions.

If TRCIDR3.STALLCTL == 0 then this field is 0.

Access to this field is **RO**.

STALLCTL, bit [26]

Indicates if trace unit implements stalling of the PE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

STALLCTL	Meaning
0b0	Stalling of the PE is not implemented.
0b1	Stalling of the PE is implemented.

Access to this field is **RO**.

SYNCPR, bit [25]

Indicates if an implementation has a fixed synchronization period.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYNCPR	Meaning
0b0	TRCSYNCPR is read/write so software can change the synchronization period.
0b1	TRCSYNCPR is read-only so the synchronization period is fixed.

This field reads as 0.

Access to this field is **RO**.

TRCERR, bit [24]

Indicates forced tracing of System Error exceptions is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCERR	Meaning
0b0	Forced tracing of System Error exceptions is not implemented.
0b1	Forced tracing of System Error exceptions is implemented.

This field reads as 1.

Access to this field is **RO**.

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]

Indicates if Non-secure EL2 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_NS_EL2	Meaning
0b0	Non-secure EL2 is not implemented.
0b1	Non-secure EL2 is implemented.

Access to this field is **RO**.

EXLEVEL_NS_EL1, bit [21]

Indicates if Non-secure EL1 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_NS_EL1	Meaning
0b0	Non-secure EL1 is not implemented.
0b1	Non-secure EL1 is implemented.

Access to this field is **RO**.

EXLEVEL_NS_EL0, bit [20]

Indicates if Non-secure EL0 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_NS_EL0	Meaning
0b0	Non-secure EL0 is not implemented.
0b1	Non-secure EL0 is implemented.

Access to this field is **RO**.

EXLEVEL_S_EL3, bit [19]

Indicates if EL3 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL3	Meaning
0b0	EL3 is not implemented.
0b1	EL3 is implemented.

Access to this field is **RO**.

EXLEVEL_S_EL2, bit [18]

Indicates if Secure EL2 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL2	Meaning
0b0	Secure EL2 is not implemented.
0b1	Secure EL2 is implemented.

Access to this field is **RO**.

EXLEVEL_S_EL1, bit [17]

Indicates if Secure EL1 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL1	Meaning
0b0	Secure EL1 is not implemented.
0b1	Secure EL1 is implemented.

Access to this field is **RO**.

EXLEVEL_S_EL0, bit [16]

Indicates if Secure EL0 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL0	Meaning
0b0	Secure EL0 is not implemented.
0b1	Secure EL0 is implemented.

Access to this field is **RO**.

Bits [15:14]

Reserved, RES0.

CCITMIN, bits [11:0]

When TRCIDR0.TRCCCI == 0:

Indicates the minimum value that can be programmed in TRCCCCTLR.THRESHOLD.

Reads as 0x000.

Access to this field is **RO**.

When TRCIDR0.TRCCCI == 1:

Indicates the minimum value that can be programmed in TRCCCCTLR.THRESHOLD.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCITMIN	Meaning
0x001..0xFFF	The minimum value that can be programmed in TRCCCCTLR.THRESHOLD.

The minimum value of this field is 0x001.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR3

TRCIDR3 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x1EC	TRCIDR3

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR4, Trace ID Register 4

The TRCIDR4 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

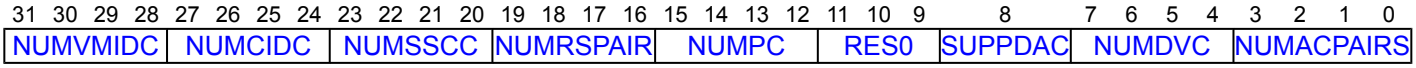
External register TRCIDR4 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR4[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR4 are RES0.

Attributes

TRCIDR4 is a 32-bit register.

Field descriptions



NUMVMIDC, bits [31:28]

Indicates the number of Virtual Context Identifier Comparators that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMVMIDC	Meaning
0b0000..0b1000	The number of Virtual Context Identifier Comparators in this implementation.

All other values are reserved.

If the PE does not implement EL2 then this field is 0b0000.

Access to this field is **RO**.

NUMCIDC, bits [27:24]

Indicates the number of Context Identifier Comparators that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMCIDC	Meaning
0b0000..0b1000	The number of Context Identifier Comparators in this implementation.

All other values are reserved.

Access to this field is **RO**.

NUMSSCC, bits [23:20]

Indicates the number of Single-shot Comparator Controls that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMSSCC	Meaning
0b0000..0b1000	The number of Single-shot Comparator Controls in this implementation.

All other values are reserved.

Access to this field is **RO**.

NUMRSPAIR, bits [19:16]

Indicates the number of resource selector pairs that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMRSPAIR	Meaning
0b0000	This implementation has zero resource selector pairs.
0b0001..0b1111	The number of resource selector pairs in this implementation, minus one.

All other values are reserved.

Access to this field is **RO**.

NUMPC, bits [15:12]

Indicates the number of PE Comparator Inputs that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMPC	Meaning
0b0000..0b1000	The number of PE Comparator Inputs in this implementation.

All other values are reserved.

Access to this field is **RO**.

Bits [11:9]

Reserved, RES0.

SUPPDAC, bit [8]

When TRCIDR4.NUMACPAIRS != 0b0000:

Indicates whether data address comparisons are implemented. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SUPPDAC	Meaning
0b0	Data address comparisons not implemented.
0b1	Data address comparisons implemented.

This field reads as 0b0.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

NUMDVC, bits [7:4]

Indicates the number of data value comparators. Data value comparators are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMDVC	Meaning
0b0000..0b1000	The number of data value comparators in this implementation.

All other values are reserved.

This field reads as 0b0000.

Access to this field is **RO**.

NUMACPAIRS, bits [3:0]

Indicates the number of Address Comparator pairs that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMACPAIRS	Meaning
0b0000..0b1000	The number of Address Comparator pairs in this implementation.

All other values are reserved.

Access to this field is **RO**.

Accessing TRCIDR4

TRCIDR4 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x1F0	TRCIDR4

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR5, Trace ID Register 5

The TRCIDR5 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

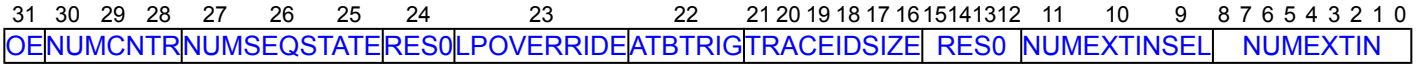
External register TRCIDR5 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR5[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR5 are RES0.

Attributes

TRCIDR5 is a 32-bit register.

Field descriptions



OE, bit [31]

Indicates support for the ETE Trace Output Enable.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OE	Meaning
0b0	ETE Trace Output Enable is not implemented.
0b1	ETE Trace Output Enable is implemented.

When FEAT_ETEv1p3 is implemented and when any IMPLEMENTATION DEFINED trace output interface is implemented, this field is 1.

Access to this field is **RO**.

NUMCNTR, bits [30:28]

Indicates the number of Counters that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMCNTR	Meaning
0b000..0b100	The number of Counters implemented.

All other values are reserved.

If TRCIDR4.NUMRSPAIR == 0b0000 then this field is 0b000.

Access to this field is **RO**.

NUMSEQSTATE, bits [27:25]

Indicates if the Sequencer is implemented and the number of Sequencer states that are implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMSEQSTATE	Meaning
0b000	The Sequencer is not implemented.
0b100	Four Sequencer states are implemented.

All other values are reserved.

If TRCIDR4.NUMRSPAIR == 0b0000 then this field is 0b000.

Access to this field is **RO**.

Bit [24]

Reserved, RES0.

LPOVERRIDE, bit [23]

Indicates support for Low-power Override Mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LPOVERRIDE	Meaning
0b0	The trace unit does not support Low-power Override Mode.
0b1	The trace unit supports Low-power Override Mode.

Access to this field is **RO**.

ATBTRIG, bit [22]

Indicates if the implementation can support ATB triggers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ATBTRIG	Meaning
0b0	The implementation does not support ATB triggers.
0b1	The implementation supports ATB triggers.

If TRCIDR4.NUMRSPAIR == 0b0000 then this field is 0.

Access to this field is **RO**.

TRACEIDSIZE, bits [21:16]

Indicates the trace ID width.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRACEIDSIZE	Meaning
0b000000	The external trace interface is not implemented.
0b000111	The implementation supports a 7-bit trace ID.

All other values are reserved.

Note

AMBA ATB requires a 7-bit trace ID width.

Access to this field is **RO**.

Bits [15:12]

Reserved, RES0.

NUMEXTINSEL, bits [11:9]

Indicates how many External Input Selector resources are implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMEXTINSEL	Meaning
0b000..0b100	The number of External Input Selector resources implemented.

All other values are reserved.

Access to this field is **RO**.

NUMEXTIN, bits [8:0]

Indicates how many External Inputs are implemented.

NUMEXTIN	Meaning
0b111111111	Unified PMU event selection.

All other values are reserved.

Access to this field is **RO**.

Accessing TRCIDR5

TRCIDR5 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x1F4	TRCIDR5

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR6, Trace ID Register 6

The TRCIDR6 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

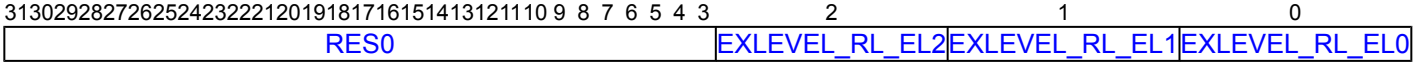
External register TRCIDR6 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR6[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR6 are RES0.

Attributes

TRCIDR6 is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [2]

Indicates if Realm EL2 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_RL_EL2	Meaning
0b0	Realm EL2 is not implemented.
0b1	Realm EL2 is implemented.

Access to this field is **RO**.

EXLEVEL_RL_EL1, bit [1]

Indicates if Realm EL1 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_RL_EL1	Meaning
0b0	Realm EL1 is not implemented.
0b1	Realm EL1 is implemented.

Access to this field is **RO**.

EXLEVEL_RL_EL0, bit [0]

Indicates if Realm EL0 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_RL_EL0	Meaning
0b0	Realm EL0 is not implemented.
0b1	Realm EL0 is implemented.

Access to this field is **RO**.

Accessing TRCIDR6

TRCIDR6 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x1F8	TRCIDR6

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR7, Trace ID Register 7

The TRCIDR7 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

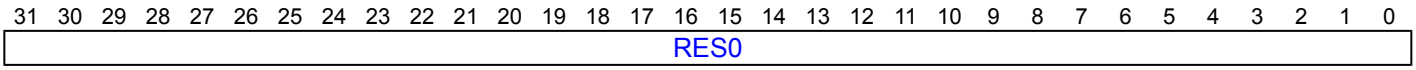
External register TRCIDR7 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR7[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR7 are RES0.

Attributes

TRCIDR7 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRCIDR7

TRCIDR7 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x1FC	TRCIDR7

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR8, Trace ID Register 8

The TRCIDR8 characteristics are:

Purpose

Returns the maximum speculation depth of the instruction trace element stream.

Configuration

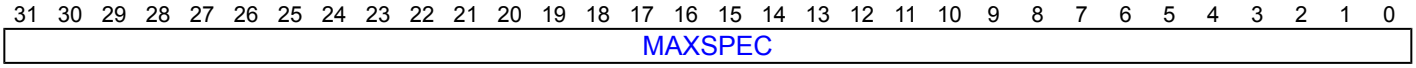
External register TRCIDR8 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR8[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR8 are RES0.

Attributes

TRCIDR8 is a 32-bit register.

Field descriptions



MAXSPEC, bits [31:0]

Indicates the maximum speculation depth of the instruction trace element stream. This is the maximum number of P0 elements in the trace element stream that can be speculative at any time.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCIDR8

TRCIDR8 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x180	TRCIDR8

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR9, Trace ID Register 9

The TRCIDR9 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

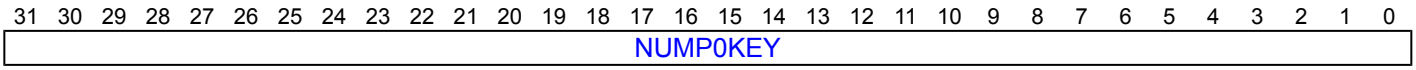
External register TRCIDR9 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR9[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIDR9 are RES0.

Attributes

TRCIDR9 is a 32-bit register.

Field descriptions



NUMP0KEY, bits [31:0]

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P0 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Accessing TRCIDR9

TRCIDR9 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x184	TRCIDR9

Accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIMSPEC0, Trace IMP DEF Register 0

The TRCIMSPEC0 characteristics are:

Purpose

TRCIMSPEC0 shows the presence of any IMPLEMENTATION DEFINED features, and provides an interface to enable the features that are provided.

Configuration

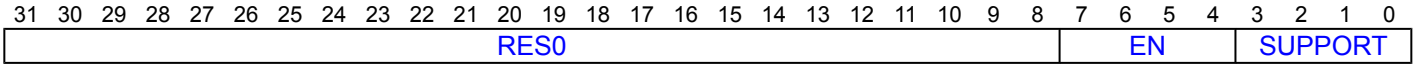
External register TRCIMSPEC0 bits [31:0] are architecturally mapped to AArch64 System register TRCIMSPEC0[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIMSPEC0 are RES0.

Attributes

TRCIMSPEC0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

EN, bits [7:4]

When TRCIMSPEC0.SUPPORT != 0b0000:

Enable. Controls whether the IMPLEMENTATION DEFINED features are enabled.

EN	Meaning
0b0000	The IMPLEMENTATION DEFINED features are not enabled. The trace unit must behave as if the IMPLEMENTATION DEFINED features are not supported.
0b0001	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0010	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0011	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0100	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0101	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0110	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0111	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1000	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1001	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1010	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1011	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1100	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1101	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1110	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1111	The trace unit behavior is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0000'.

Otherwise:

Reserved, RES0.

SUPPORT, bits [3:0]

Indicates whether the implementation supports IMPLEMENTATION DEFINED features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SUPPORT	Meaning
0b0000	No IMPLEMENTATION DEFINED features are supported.
0b0001	IMPLEMENTATION DEFINED features are supported.
0b0010	IMPLEMENTATION DEFINED features are supported.
0b0011	IMPLEMENTATION DEFINED features are supported.
0b0100	IMPLEMENTATION DEFINED features are supported.
0b0101	IMPLEMENTATION DEFINED features are supported.
0b0110	IMPLEMENTATION DEFINED features are supported.
0b0111	IMPLEMENTATION DEFINED features are supported.
0b1000	IMPLEMENTATION DEFINED features are supported.
0b1001	IMPLEMENTATION DEFINED features are supported.
0b1010	IMPLEMENTATION DEFINED features are supported.
0b1011	IMPLEMENTATION DEFINED features are supported.
0b1100	IMPLEMENTATION DEFINED features are supported.
0b1101	IMPLEMENTATION DEFINED features are supported.
0b1110	IMPLEMENTATION DEFINED features are supported.
0b1111	IMPLEMENTATION DEFINED features are supported.

Use of nonzero values requires written permission from Arm.

Access to this field is **RO**.

Accessing TRCIMSPEC0

TRCIMSPEC0 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x1C0	TRCIMSPEC0

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIMSPEC<n>, Trace IMP DEF Register <n>, n = 1 - 7

The TRCIMSPEC<n> characteristics are:

Purpose

These registers might return information that is specific to an implementation, or enable features specific to an implementation to be programmed. The product Technical Reference Manual describes these registers.

Configuration

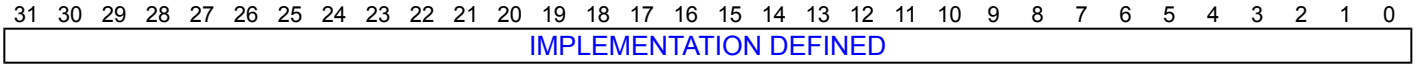
External register TRCIMSPEC<n> bits [31:0] are architecturally mapped to AArch64 System register TRCIMSPEC<n>[31:0].

This register is present only when an implementation implements TRCIMSPEC<n>, FEAT_ETE is implemented, and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCIMSPEC<n> are RES0.

Attributes

TRCIMSPEC<n> is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Accessing TRCIMSPEC<n>

TRCIMSPEC<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0x1C0` + (4 * n)	TRCIMSPEC<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCITCTRL, Trace Integration Mode Control Register

The TRCITCTRL characteristics are:

Purpose

A component can use TRCITCTRL to dynamically switch between functional mode and integration mode. In integration mode, topology detection is enabled. After switching to integration mode and performing integration tests or topology detection, reset the system to ensure correct behavior of CoreSight and other connected system components.

For additional information, see the CoreSight Architecture Specification.

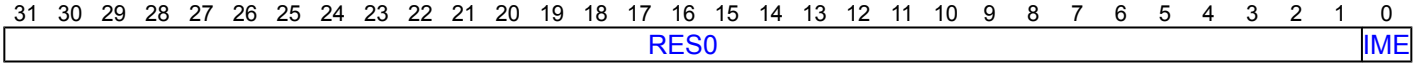
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCITCTRL are RES0.

Attributes

TRCITCTRL is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

IME, bit [0]

When topology detection or integration functionality is implemented:

Integration Mode Enable.

IME	Meaning
0b0	Component functional mode.
0b1	Component integration mode. Support for topology detection and integration testing is enabled.

Otherwise:

Reserved, RES0.

Accessing TRCITCTRL

External debugger accesses to this register are IMPLEMENTATION DEFINED when the trace unit is not in the Idle state.

TRCITCTRL can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xF00	TRCITCTRL

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCITEEDCR, Instrumentation Trace Extension External Debug Control Register

The TRCITEEDCR characteristics are:

Purpose

Controls instrumentation trace filtering.

Configuration

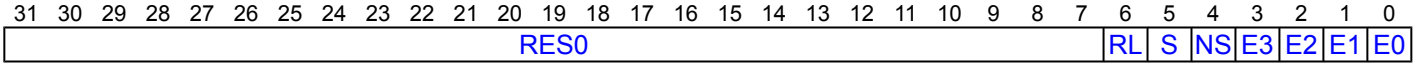
External register TRCITEEDCR bits [31:0] are architecturally mapped to AArch64 System register TRCITEEDCR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and FEAT_ITE is implemented. Otherwise, direct accesses to TRCITEEDCR are RES0.

Attributes

TRCITEEDCR is a 32-bit register.

Field descriptions



Bits [31:7]

Reserved, RES0.

RL, bit [6]

When FEAT_RME is implemented:

Instrumentation Trace in Realm state.

RL	Meaning
0b0	Instrumentation trace prohibited in Realm state.
0b1	Instrumentation trace permitted in Realm state.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

This field is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S, bit [5]

When Secure state is implemented:

Instrumentation Trace in Secure state.

S	Meaning
0b0	Instrumentation trace prohibited in Secure state.
0b1	Instrumentation trace permitted in Secure state.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

When FEAT_RME is not implemented, this field is used in conjunction with TRCCONFIGR.ITO, TRCITEEDCR.E3, and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Secure state.

When FEAT_RME is implemented, this field is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NS, bit [4]

When Non-secure state is implemented:

Instrumentation Trace in Non-secure state.

NS	Meaning
0b0	Instrumentation trace prohibited in Non-secure state.
0b1	Instrumentation trace permitted in Non-secure state.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

This field is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E3, bit [3]

When EL3 is implemented:

Instrumentation Trace Enable at EL3.

E3	Meaning
0b0	Instrumentation trace prohibited at EL3.
0b1	Instrumentation trace permitted at EL3.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

When FEAT_RME is not implemented, TRCITEEDCR.E3 is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.S to control whether Instrumentation trace is permitted or prohibited at EL3.

When FEAT_RME is implemented, TRCITEEDCR.E3 is used in conjunction with TRCCONFIGR.ITO to control whether Instrumentation trace is permitted or prohibited at EL3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E<m>, bit [m], for m = 2 to 0

Instrumentation Trace Enable at EL<m>.

E<m>	Meaning
0b0	Instrumentation trace prohibited at EL<m>.
0b1	Instrumentation trace permitted at EL<m>.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

This bit is used in conjunction with TRCCONFIGR.ITO, TRCITEEDCR.NS, TRCITEEDCR.S, and TRCITEEDCR.RL to control whether Instrumentation trace is permitted or prohibited at EL<m> in the specified Security states.

TRCITEEDCR.E<2> is RES0 if EL2 is not implemented in any Security states.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCITEEDCR

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCITEEDCR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x048	TRCITEEDCR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCLAR, Trace Lock Access Register

The TRCLAR characteristics are:

Purpose

Used to lock and unlock the Software Lock.

Note

ETE does not implement the Software Lock.

For additional information, see the CoreSight Architecture Specification.

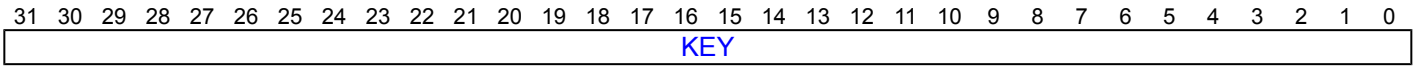
Configuration

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and ETE Software Lock is implemented. Otherwise, direct accesses to TRCLAR are RES0.

Attributes

TRCLAR is a 32-bit register.

Field descriptions



KEY, bits [31:0]

When ETE Software Lock is implemented:

Software Lock Key.

A value of `0xC5ACCE55` unlocks the Software Lock.

Any other value locks the Software Lock.

Otherwise:

Reserved, RES0.

Accessing TRCLAR

External debugger accesses to this register are unaffected by the OS Lock.

TRCLAR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0xFB0`	TRCLAR

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.

- Otherwise, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCLSR, Trace Lock Status Register

The TRCLSR characteristics are:

Purpose

Indicates whether the Software Lock is implemented, and the current status of the Software Lock.

For additional information, see the CoreSight Architecture Specification.

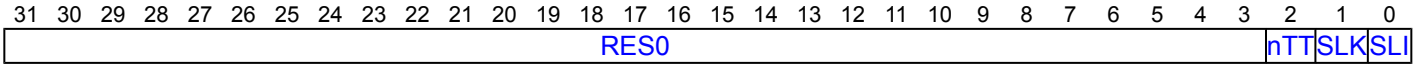
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCLSR are RES0 .

Attributes

TRCLSR is a 32-bit register.

Field descriptions



Bits [31:3]

Reserved, RES0.

nTT, bit [2]

Software lock size.

Reads as `0b0`.

Access to this field is **RO**.

SLK, bit [1]

The current Software Lock status.

SLK	Meaning
`0b0`	Software Lock is unlocked.
`0b1`	Software Lock is locked. Writes to the other registers in this component, except for the TRCLAR, are ignored.

This field reads as 0.

SLI, bit [0]

Indicates whether the Software Lock is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SLI	Meaning
`0b0`	Software Lock is not implemented. Writes to the TRCLAR are ignored.
`0b1`	Software Lock is implemented.

This field reads as 0.

Access to this field is **RO**.

Accessing TRCLSR

External debugger accesses to this register are unaffected by the OS Lock.

TRCLSR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFB4	TRCLSR

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCOSLSR, Trace OS Lock Status Register

The TRCOSLSR characteristics are:

Purpose

Returns the status of the Trace OS Lock.

Configuration

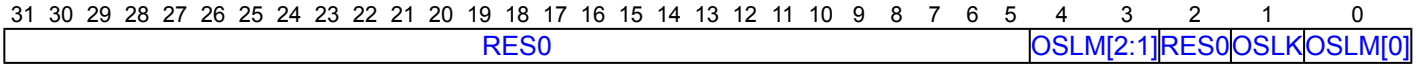
External register TRCOSLSR bits [31:0] are architecturally mapped to AArch64 System register TRCOSLSR[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCOSLSR are RES0.

Attributes

TRCOSLSR is a 32-bit register.

Field descriptions



Bits [31:5]

Reserved, RES0.

OSLM, bits [4:3, 0]

OS Lock model.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OSLM	Meaning
0b000	Trace OS Lock is not implemented.
0b010	Trace OS Lock is implemented.
0b100	Trace OS Lock is not implemented, and the trace unit is controlled by the PE OS Lock.

All other values are reserved.

When FEAT_ETE is implemented, the values 0b000 and 0b010 are not permitted.

The OSLM field is split as follows:

- OSLM[2:1] is TRCOSLSR[4:3].
- OSLM[0] is TRCOSLSR[0].

Access to this field is **RO**.

Bit [2]

Reserved, RES0.

OSLK, bit [1]

OS Lock status.

OSLK	Meaning
0b0	The OS Lock is unlocked.
0b1	The OS Lock is locked.

When FEAT_ETE is implemented, this field indicates the state of the PE OS Lock.

When FEAT_ETMv4 is implemented, this field indicates the state of the Trace OS Lock.

Accessing TRCOSLSR

External debugger accesses to this register are unaffected by the OS Lock.

TRCOSLSR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x304	TRCOSLSR

Accessible as follows:

- When !AllowExternalTraceAccess(addrdesc) or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPDCR, Trace PowerDown Control Register

The TRCPDCR characteristics are:

Purpose

Requests the system to provide power to the trace unit.

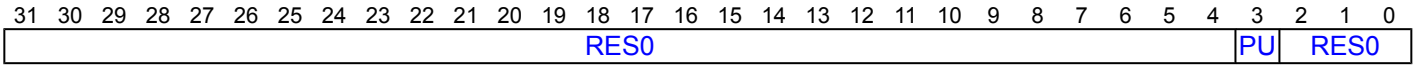
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPDCR are RES0.

Attributes

TRCPDCR is a 32-bit register.

Field descriptions



Bits [31:4]

Reserved, RES0.

PU, bit [3]

Power Up Request.

PU	Meaning
0b0	The system can remove power from the trace unit core power domain, or requests for power to the trace unit core power domain are implemented outside of the trace unit.
0b1	The system must provide power to the trace unit core power domain.

This field is RES0.

Bits [2:0]

Reserved, RES0.

Accessing TRCPDCR

External debugger accesses to this register are unaffected by the OS Lock.

TRCPDCR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x310	TRCPDCR

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.

- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPDSR, Trace PowerDown Status Register

The TRCPDSR characteristics are:

Purpose

Indicates the power status of the trace unit.

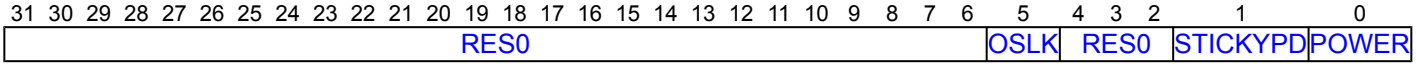
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPDSR are RES0.

Attributes

TRCPDSR is a 32-bit register.

Field descriptions



Bits [31:6]

Reserved, RES0.

OSLK, bit [5]

OS Lock Status.

OSLK	Meaning
0b0	The OS Lock is unlocked.
0b1	The OS Lock is locked.

Note

This field indicates the state of the PE OS Lock.

Bits [4:2]

Reserved, RES0.

STICKYPD, bit [1]

Sticky powerdown status. Indicates whether the trace register state is valid.

STICKYPD	Meaning
0b0	The state of TRCOSLSR and the trace registers are valid.
0b1	The state of TRCOSLSR and the trace registers might not be valid.

This field is set to 1 if the power to the trace unit core power domain is removed and the trace unit register state is not valid.

The STICKYPD field is read-sensitive. On a read of the TRCPDSR, this field is cleared to 0 after the register has been read.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RC/WI**.

POWER, bit [0]

Power Status.

POWER	Meaning
0b0	The trace unit core power domain is not powered. All trace unit registers are not accessible and they all return an error response.
0b1	The trace unit core power domain is powered. Trace unit registers are accessible.

Access to this field is **RAO/WI**.

Accessing TRCPDSR

External debugger accesses to this register are unaffected by the OS Lock.

TRCPDSR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x314	TRCPDSR

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR0, Trace Peripheral Identification Register 0

The TRCPIDR0 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

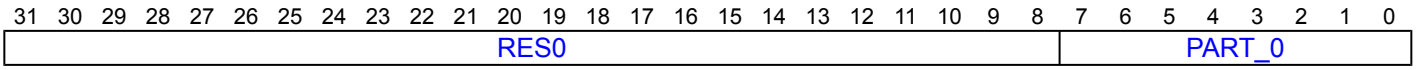
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPIDR0 are RES0.

Attributes

TRCPIDR0 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, bits [7:0].

The part number is selected by the designer of the component, and is stored in TRCPIDR1.PART_1 and TRCPIDR0.PART_0.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCPIDR0

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR0 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFE0	TRCPIDR0

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR1, Trace Peripheral Identification Register 1

The TRCPIDR1 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

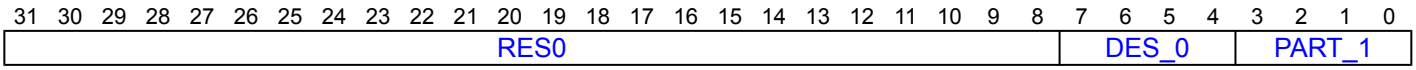
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPIDR1 are RES0.

Attributes

TRCPIDR1 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, JEP106 identification code, bits [3:0].

JEP106 identification and continuation codes, which are stored as follows:

- TRCPIDR1.DES_0: JEP106 identification code bits[3:0].
- TRCPIDR2.DES_1: JEP106 identification code bits[6:4].
- TRCPIDR4.DES_2: JEP106 continuation code.

These codes indicate the designer of the component and not the implementer, except where the two are the same. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

A JEDEC code takes the following form:

- A sequence of zero or more numbers, all having the value `0x7F`.
- A following 8-bit number, that is not `0x7F`, and where bit[7] is an odd parity bit.

The parity bit in the JEP106 identification code is not included.

Note

For example, Arm Limited is assigned the code `0x7F 0x7F 0x7F 0x7F 0x3B`.

- The continuation code is the number of times `0x7F` appears before the final number. For example, a component designed by Arm Limited has the code `0x4`.
- The identification code is bits[6:0] of the final number. For example, a component designed by Arm Limited has the code `0x3B`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, bits [11:8].

The part number is selected by the designer of the component, and is stored in TRCPIDR1.PART_1 and TRCPIDR0.PART_0.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCPIDR1

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR1 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFE4	TRCPIDR1

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR2, Trace Peripheral Identification Register 2

The TRCPIDR2 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

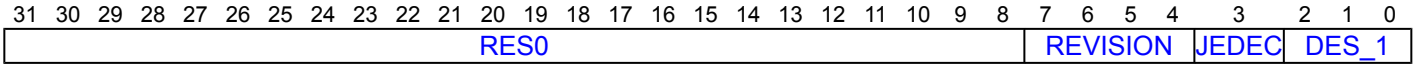
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPIDR2 are RES0.

Attributes

TRCPIDR2 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Component major revision.

TRCPIDR2.REVISION and TRCPIDR3.REVAND together form the revision number of the component, with TRCPIDR2.REVISION being the most significant part and TRCPIDR3.REVAND the least significant part.

When a component is changed, TRCPIDR2.REVISION or TRCPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. TRCPIDR3.REVAND should be set to `0b0000` when TRCPIDR2.REVISION is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as `0b1`.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, JEP106 identification code, bits [6:4].

JEP106 identification and continuation codes, which are stored as follows:

- TRCPIDR1.DES_0: JEP106 identification code bits[3:0].

- TRCPIDR2.DES_1: JEP106 identification code bits[6:4].
- TRCPIDR4.DES_2: JEP106 continuation code.

These codes indicate the designer of the component and not the implementer, except where the two are the same. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

A JEDEC code takes the following form:

- A sequence of zero or more numbers, all having the value `0x7F`.
- A following 8-bit number, that is not `0x7F`, and where bit[7] is an odd parity bit.

The parity bit in the JEP106 identification code is not included.

Note

For example, Arm Limited is assigned the code `0x7F 0x7F 0x7F 0x7F 0x3B`.

- The continuation code is the number of times `0x7F` appears before the final number. For example, a component designed by Arm Limited has the code `0x4`.
- The identification code is bits[6:0] of the final number. For example, a component designed by Arm Limited has the code `0x3B`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCPIDR2

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR2 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0xFE8`	TRCPIDR2

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR3, Trace Peripheral Identification Register 3

The TRCPIDR3 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

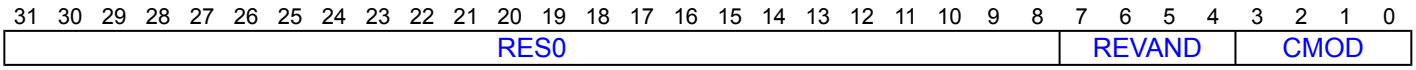
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPIDR3 are RES0.

Attributes

TRCPIDR3 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Component minor revision.

TRCPIDR2.REVISION and TRCPIDR3.REVAND together form the revision number of the component, with TRCPIDR2.REVISION being the most significant part and TRCPIDR3.REVAND the least significant part. When a component is changed, TRCPIDR2.REVISION or TRCPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. TRCPIDR3.REVAND should be set to `0b0000` when TRCPIDR2.REVISION is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer Modified.

Indicates the component has been modified.

A value of `0b0000` means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components equals zero, the components are identical.
- If the CMOD fields of both components have the same nonzero value, it does not necessarily mean that they have the same modifications.
- If the value of the CMOD field of either of the two components is nonzero, they might not be identical, even though they have the same Unique Component Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCPIDR3

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR3 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFEC	TRCPIDR3

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR4, Trace Peripheral Identification Register 4

The TRCPIDR4 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

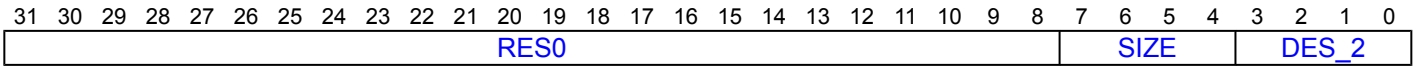
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPIDR4 are RES0.

Attributes

TRCPIDR4 is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIZE	Meaning
0b0000	One of the following is true: • The component uses a single 4KB block. • The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.
0b0001..0b1111	The component occupies $2^{\text{TRCPIDR4.SIZE}}$ 4KB blocks.

Using this field to indicate the size of the component is deprecated. This field might not correctly indicate the size of the component. Arm recommends that software determine the size of the component from the Unique Component Identifier fields, and other IMPLEMENTATION DEFINED registers in the component.

This field has the value 0b0000.

Access to this field is **RO**.

DES_2, bits [3:0]

Designer, JEP106 continuation code.

JEP106 identification and continuation codes, which are stored as follows:

- TRCPIDR1.DES_0: JEP106 identification code bits[3:0].
- TRCPIDR2.DES_1: JEP106 identification code bits[6:4].
- TRCPIDR4.DES_2: JEP106 continuation code.

These codes indicate the designer of the component and not the implementer, except where the two are the same. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

A JEDEC code takes the following form:

- A sequence of zero or more numbers, all having the value `0x7F`.
- A following 8-bit number, that is not `0x7F`, and where bit[7] is an odd parity bit.

The parity bit in the JEP106 identification code is not included.

Note

For example, Arm Limited is assigned the code `0x7F 0x7F 0x7F 0x7F 0x3B`.

- The continuation code is the number of times `0x7F` appears before the final number. For example, a component designed by Arm Limited has the code `0x4`.
- The identification code is bits[6:0] of the final number. For example, a component designed by Arm Limited has the code `0x3B`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCPIDR4

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR4 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0xFD0`	TRCPIDR4

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR5, Trace Peripheral Identification Register 5

The TRCPIDR5 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

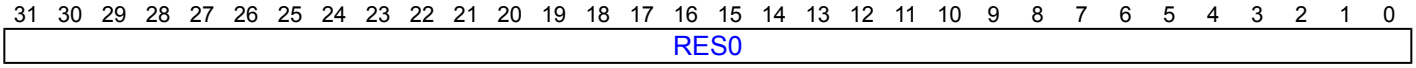
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPIDR5 are RES0.

Attributes

TRCPIDR5 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRCPIDR5

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR5 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFD4	TRCPIDR5

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR6, Trace Peripheral Identification Register 6

The TRCPIDR6 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

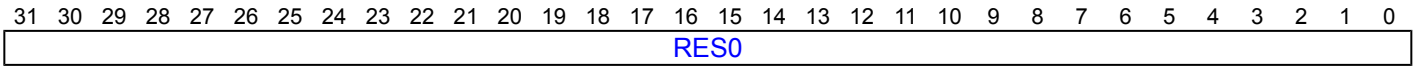
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPIDR6 are RES0.

Attributes

TRCPIDR6 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRCPIDR6

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR6 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFD8	TRCPIDR6

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR7, Trace Peripheral Identification Register 7

The TRCPIDR7 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

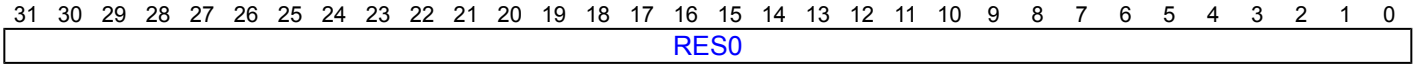
Configuration

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPIDR7 are RES0.

Attributes

TRCPIDR7 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRCPIDR7

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR7 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0xFDC	TRCPIDR7

Accessible as follows:

- When !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPRGCTLR, Trace Programming Control Register

The TRCPRGCTLR characteristics are:

Purpose

Enables the trace unit.

Configuration

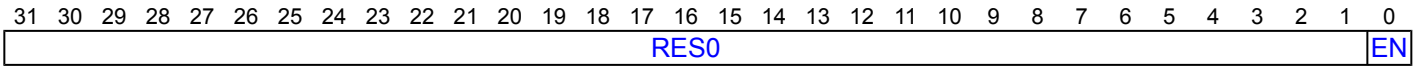
External register TRCPRGCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCPRGCTLR[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCPRGCTLR are RES0.

Attributes

TRCPRGCTLR is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

EN, bit [0]

Trace unit enable.

EN	Meaning
0b0	The trace unit is disabled.
0b1	The trace unit is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0'.

Accessing TRCPRGCTLR

Must be programmed.

TRCPRGCTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x004	TRCPRGCTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCQCTLR, Trace Q Element Control Register

The TRCQCTLR characteristics are:

Purpose

Controls when Q elements are enabled.

Configuration

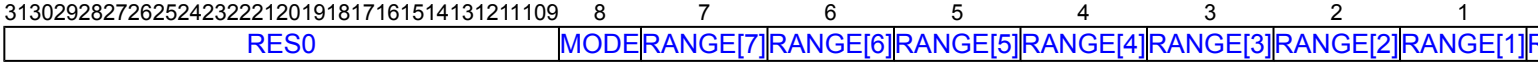
External register TRCQCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCQCTLR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and TRCIDR0.QFILT == 1. Otherwise, direct accesses to TRCQCTLR are RES0.

Attributes

TRCQCTLR is a 32-bit register.

Field descriptions



Bits [31:9]

Reserved, RES0.

MODE, bit [8]

Selects whether the Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit is permitted to generate Q elements or address ranges where the trace unit is not permitted to generate Q elements:

MODE	Meaning
0b0	Exclude mode. The Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit must not generate Q elements. If no ranges are selected, Q elements are permitted across the entire memory map.
0b1	Include Mode. The Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit can generate Q elements. If all the implemented bits in RANGE are set to 0 then Q elements are disabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE[<m>], bit [m], for m = 7 to 0

Specifies whether Address Range Comparator <m> controls Q elements.

RANGE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines is not selected.
0b1	The address range that Address Range Comparator <m> defines is selected.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCQCTLR

Must be programmed if TRCCONFIGR.QE != `0b00`.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCQCTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0x044`	TRCQCTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCRSCTLR<n>, Trace Resource Selection Control Register <n>, n = 2 - 31

The TRCRSCTLR<n> characteristics are:

Purpose

Controls the selection of the resources in the trace unit.

Configuration

External register TRCRSCTLR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCRSCTLR<n>[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and (UInt(TRCIDR4.NUMRSPAIR) + 1) * 2 > n. Otherwise, direct accesses to TRCRSCTLR<n> are RES0.

Resource selector 0 always returns FALSE.

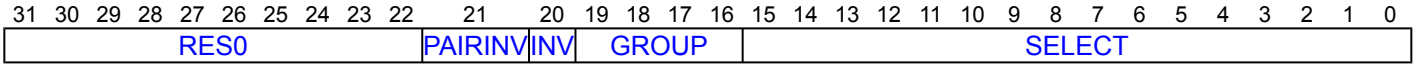
Resource selector 1 always returns TRUE.

Resource selectors are implemented in pairs. Each odd numbered resource selector is part of a pair with the even numbered resource selector that is numbered as one less than it. For example, resource selectors 2 and 3 form a pair.

Attributes

TRCRSCTLR<n> is a 32-bit register.

Field descriptions



Bits [31:22]

Reserved, RES0.

PAIRINV, bit [21]

When n is even:

Controls whether the combined result from a resource selector pair is inverted.

PAIRINV	Meaning
0b0	Do not invert the combined output of the 2 resource selectors.
0b1	Invert the combined output of the 2 resource selectors.

If:

- A is the register TRCRSCTLR<n>.
- B is the register TRCRSCTLR<n+1>.

Then the combined output of the 2 resource selectors A and B depends on the value of (A.PAIRINV, A.INV, B.INV) as follows:

- 0b000 -> A and B.
- 0b001 -> Reserved.
- 0b010 -> not(A) and B.
- 0b011 -> not(A) and not(B).
- 0b100 -> not(A) or not(B).

- `0b101` -> not(A) or B.
- `0b110` -> Reserved.
- `0b111` -> A or B.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

INV, bit [20]

Controls whether the resource, that TRCRSCTLR<n>.GROUP and TRCRSCTLR<n>.SELECT selects, is inverted.

INV	Meaning
`0b0`	Do not invert the output of this selector.
`0b1`	Invert the output of this selector.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

GROUP, bits [19:16]

Selects a group of resources.

GROUP	Meaning	SELECT
`0b0000`	External Input Selectors.	SELECT encoding for External Input Selectors
`0b0001`	PE Comparator Inputs.	SELECT encoding for PE Comparator Inputs
`0b0010`	Counters and Sequencer.	SELECT encoding for Counters and Sequencer
`0b0011`	Single-shot Comparator Controls.	SELECT encoding for Single-shot Comparator Controls
`0b0100`	Single Address Comparators.	SELECT encoding for Single Address Comparators
`0b0101`	Address Range Comparators.	SELECT encoding for Address Range Comparators
`0b0110`	Context Identifier Comparators.	SELECT encoding for Context Identifier Comparators
`0b0111`	Virtual Context Identifier Comparators.	SELECT encoding for Virtual Context Identifier Comparators

All other values are reserved.

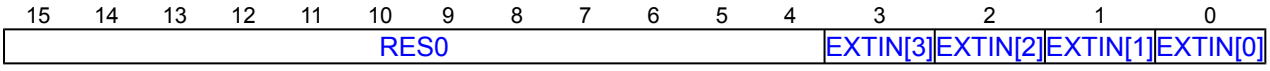
The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT, bits [15:0]

Resource Specific Controls. Contains the controls specific to the resource group selected by GROUP, described in the following sections.

SELECT encoding for External Input Selectors



Bits [15:4]

Reserved, RES0.

EXTIN[<m>], bit [m], for m = 3 to 0

Selects one or more External Inputs.

EXTIN[<m>]	Meaning
0b0	Ignore EXTIN <m>.
0b1	Select EXTIN <m>.

This bit is RES0 if m >= TRCIDR5.NUMEXTINSEL.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for PE Comparator Inputs

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0								PECOMP[7]	PECOMP[6]	PECOMP[5]	PECOMP[4]	PECOMP[3]	PECOMP[2]	PECOMP[1]	PECOMP[0]

Bits [15:8]

Reserved, RES0.

PECOMP[<m>], bit [m], for m = 7 to 0

Selects one or more PE Comparator Inputs.

PECOMP[<m>]	Meaning
0b0	Ignore PE Comparator Input <m>.
0b1	Select PE Comparator Input <m>.

This bit is RES0 if m >= TRCIDR4.NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Counters and Sequencer

15	14	13	12	11	10	9	8	7	6	5	4	3	2
RES0								SEQUENCER[3]	SEQUENCER[2]	SEQUENCER[1]	SEQUENCER[0]	COUNTERS[3]	COUNTERS[2]	COUN

Bits [15:8]

Reserved, RES0.

SEQUENCER[<m>], bit [m+4], for m = 3 to 0

Sequencer states.

SEQUENCER[<m>]	Meaning
0b0	Ignore Sequencer state <m>.
0b1	Select Sequencer state <m>.

This bit is RES0 if m >= TRCIDR5.NUMSEQSTATE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

COUNTERS[<m>], bit [m], for m = 3 to 0

Counters resources at zero.

COUNTERS[<m>]	Meaning
0b0	Ignore Counter <m>.
0b1	Select Counter <m> is zero.

This bit is RES0 if m >= TRCIDR5.NUMCNTR.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single-shot Comparator Controls

15	14	13	12	11	10	9	8	7	6	5	4	3	2
RES0								SINGLE_SHOT[7]	SINGLE_SHOT[6]	SINGLE_SHOT[5]	SINGLE_SHOT[4]	SINGLE_SHOT[3]	SINGLE_SH

Bits [15:8]

Reserved, RES0.

SINGLE_SHOT[<m>], bit [m], for m = 7 to 0

Selects one or more Single-shot Comparator Controls.

SINGLE_SHOT[<m>]	Meaning
0b0	Ignore Single-shot Comparator Control <m>.
0b1	Select Single-shot Comparator Control <m>.

This bit is RES0 if m >= TRCIDR4.NUMSSCC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single Address Comparators

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
SAC[15]	SAC[14]	SAC[13]	SAC[12]	SAC[11]	SAC[10]	SAC[9]	SAC[8]	SAC[7]	SAC[6]	SAC[5]	SAC[4]	SAC[3]	SAC[2]	SAC[1]	SA

SAC[<m>], bit [m], for m = 15 to 0

Selects one or more Single Address Comparators.

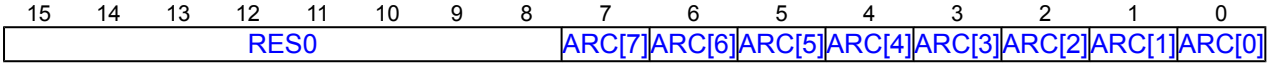
SAC[<m>]	Meaning
0b0	Ignore Single Address Comparator <m>.
0b1	Select Single Address Comparator <m>.

This bit is RES0 if m >= 2 × TRCIDR4.NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Address Range Comparators



Bits [15:8]

Reserved, RES0.

ARC[<m>], bit [m], for m = 7 to 0

Selects one or more Address Range Comparators.

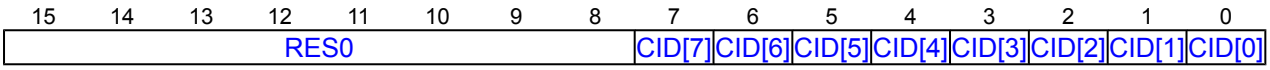
ARC[<m>]	Meaning
0b0	Ignore Address Range Comparator <m>.
0b1	Select Address Range Comparator <m>.

This bit is RES0 if m >= TRCIDR4.NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Context Identifier Comparators



Bits [15:8]

Reserved, RES0.

CID[<m>], bit [m], for m = 7 to 0

Selects one or more Context Identifier Comparators.

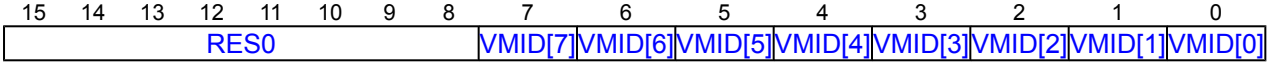
CID[<m>]	Meaning
0b0	Ignore Context Identifier Comparator <m>.
0b1	Select Context Identifier Comparator <m>.

This bit is RES0 if m >= TRCIDR4.NUMCIDC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Virtual Context Identifier Comparators



Bits [15:8]

Reserved, RES0.

VMID[<m>], bit [m], for m = 7 to 0

Selects one or more Virtual Context Identifier Comparators.

VMID[<m>]	Meaning
0b0	Ignore Virtual Context Identifier Comparator <m>.
0b1	Select Virtual Context Identifier Comparator <m>.

This bit is RES0 if m >= TRCIDR4.NUMVMIDC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCRSCTLR<n>

Must be programmed if any of the following are true:

- TRCCNTCTLR<a>.RLDEVENT.TYPE == 0 and TRCCNTCTLR<a>.RLDEVENT.SEL == n.
- TRCCNTCTLR<a>.RLDEVENT.TYPE == 1 and TRCCNTCTLR<a>.RLDEVENT.SEL == n/2.
- TRCCNTCTLR<a>.CNTEVENT.TYPE == 0 and TRCCNTCTLR<a>.CNTEVENT.SEL == n.
- TRCCNTCTLR<a>.CNTEVENT.TYPE == 1 and TRCCNTCTLR<a>.CNTEVENT.SEL == n/2.
- TRCEVENTCTL0R.EVENT0.TYPE == 0 and TRCEVENTCTL0R.EVENT0.SEL == n.
- TRCEVENTCTL0R.EVENT0.TYPE == 1 and TRCEVENTCTL0R.EVENT0.SEL == n/2.
- TRCEVENTCTL0R.EVENT1.TYPE == 0 and TRCEVENTCTL0R.EVENT1.SEL == n.
- TRCEVENTCTL0R.EVENT1.TYPE == 1 and TRCEVENTCTL0R.EVENT1.SEL == n/2.
- TRCEVENTCTL0R.EVENT2.TYPE == 0 and TRCEVENTCTL0R.EVENT2.SEL == n.
- TRCEVENTCTL0R.EVENT2.TYPE == 1 and TRCEVENTCTL0R.EVENT2.SEL == n/2.
- TRCEVENTCTL0R.EVENT3.TYPE == 0 and TRCEVENTCTL0R.EVENT3.SEL == n.
- TRCEVENTCTL0R.EVENT3.TYPE == 1 and TRCEVENTCTL0R.EVENT3.SEL == n/2.
- TRCSEQEVR<a>.B.TYPE == 0 and TRCSEQEVR<a>.B.SEL = n.
- TRCSEQEVR<a>.B.TYPE == 1 and TRCSEQEVR<a>.B.SEL = n/2.
- TRCSEQEVR<a>.F.TYPE == 0 and TRCSEQEVR<a>.F.SEL = n.
- TRCSEQEVR<a>.F.TYPE == 1 and TRCSEQEVR<a>.F.SEL = n/2.
- TRCSEQRSTEVR.RST.TYPE == 0 and TRCSEQRSTEVR.RST.SEL == n.
- TRCSEQRSTEVR.RST.TYPE == 1 and TRCSEQRSTEVR.RST.SEL == n/2.
- TRCTSCTLR.EVENT.TYPE == 0 and TRCTSCTLR.EVENT.SEL == n.
- TRCTSCTLR.EVENT.TYPE == 1 and TRCTSCTLR.EVENT.SEL == n/2.
- TRCVICTLR.EVENT.TYPE == 0 and TRCVICTLR.EVENT.SEL == n.
- TRCVICTLR.EVENT.TYPE == 1 and TRCVICTLR.EVENT.SEL == n/2.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCRSCTLR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x200 + (4 * n)	TRCRSCTLR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCRSR, Trace Resources Status Register

The TRCRSR characteristics are:

Purpose

Use this to set, or read, the status of the resources.

Configuration

External register TRCRSR bits [31:0] are architecturally mapped to AArch64 System register TRCRSR[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCRSR are RES0 .

Attributes

TRCRSR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0																			TA	EVENT[3]	EVENT[2]	EVENT[1]	EVENT[0]	RES0				EXTIN[3]	EXTIN[2]	EXTIN[1]	EXTIN[0]

Bits [31:13]

Reserved, RES0.

TA, bit [12]

Tracing active.

TA	Meaning
0b0	Tracing is not active.
0b1	Tracing is active.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

EVENT[<m>], bit [m+8], for m = 3 to 0

Untraced status of ETEEvents.

EVENT[<m>]	Meaning
0b0	An ETEEvent <m> has not occurred.
0b1	An ETEEvent <m> has occurred while the resources were in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When TRCIDR4.NUMRSPAIR == 0b0000, access to this field is **RES0**.
- Access to this field is **RES0** if all the following are true:
  - TRCIDR4.NUMRSPAIR != 0b0000.

        - m > UInt(TRCIDR0.NUMEVENT).
    - Otherwise, access to this field is **RW**.

Bits [7:4]

Reserved, RES0.

EXTIN[<m>], bit [m], for m = 3 to 0

The sticky status of the External Input Selectors.

EXTIN[<m>]	Meaning
0b0	An event selected by External Input Selector <m> has not occurred.
0b1	At least one event selected by External Input Selector <m> has occurred while the resources were in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR5.NUMEXTINSEL), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCRSR

Must always be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCRSR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x028	TRCRSR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQEVR<n>, Trace Sequencer State Transition Control Register <n>, n = 0 - 2

The TRCSEQEVR<n> characteristics are:

Purpose

Moves the Sequencer state:

- Backwards, from state n+1 to state n when a programmed resource event occurs.
- Forwards, from state n to state n+1 when a programmed resource event occurs.

Configuration

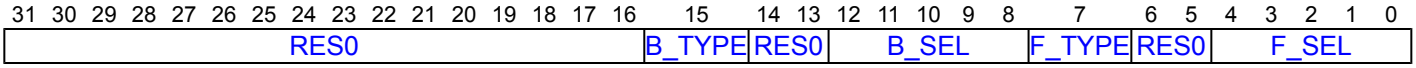
External register TRCSEQEVR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCSEQEVR<n>[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQEVR<n> are RES0.

Attributes

TRCSEQEVR<n> is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

B_TYPE, bit [15]

Backward field. Selects an event that causes the Sequencer to move from state n+1 to state n. For example, if TRCSEQEVR2.B.SEL == `0x14` then when event `0x14` occurs, the Sequencer moves from state 3 to state 2.

Chooses the type of Resource Selector.

B_TYPE	Meaning
`0b0`	A single Resource Selector. TRCSEQEVR<n>.B.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
`0b1`	A Boolean-combined pair of Resource Selectors. TRCSEQEVR<n>.B.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR<n>.B.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

B_SEL, bits [12:8]

Backward field. Selects an event that causes the Sequencer to move from state n+1 to state n. For example, if TRCSEQEVR2.B.SEL == `0x14` then when event `0x14` occurs, the Sequencer moves from state 3 to state 2.

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.B.TYPE controls whether TRCSEQEVR<n>.B.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

F_TYPE, bit [7]

Forward field. Selects an event that causes the Sequencer to move from state n to state n+1. For example, if TRCSEQEVR1.F.SEL == `0x12` then when event `0x12` occurs, the Sequencer moves from state 1 to state 2.

Chooses the type of Resource Selector.

F_TYPE	Meaning
`0b0`	A single Resource Selector. TRCSEQEVR<n>.F.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
`0b1`	A Boolean-combined pair of Resource Selectors. TRCSEQEVR<n>.F.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR<n>.F.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

F_SEL, bits [4:0]

Forward field. Selects an event that causes the Sequencer to move from state n to state n+1. For example, if TRCSEQEVR1.F.SEL == `0x12` then when event `0x12` occurs, the Sequencer moves from state 1 to state 2.

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.F.TYPE controls whether TRCSEQEVR<n>.F.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQEVR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.SEQUENCER != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSEQEVR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x100 + (4 * n)	TRCSEQEVR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQRSTEVR, Trace Sequencer Reset Control Register

The TRCSEQRSTEVR characteristics are:

Purpose

Moves the Sequencer to state 0 when a programmed resource event occurs.

Configuration

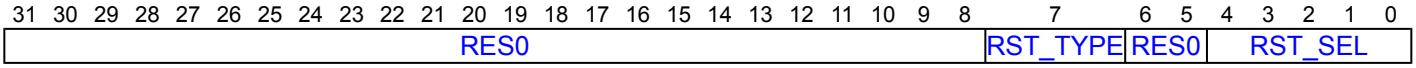
External register TRCSEQRSTEVR bits [31:0] are architecturally mapped to AArch64 System register TRCSEQRSTEVR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQRSTEVR are RES0.

Attributes

TRCSEQRSTEVR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

RST_TYPE, bit [7]

Reset field. Selects an event that causes the Sequencer to move to state 0.

Chooses the type of Resource Selector.

RST_TYPE	Meaning
0b0	A single Resource Selector. TRCSEQRSTEVR.RST.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCSEQRSTEVR.RST.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQRSTEVR.RST.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

RST_SEL, bits [4:0]

Reset field. Selects an event that causes the Sequencer to move to state 0.

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQRSTEVR.RST.TYPE controls whether TRCSEQRSTEVR.RST.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQRSTEVR

Must be programmed if TRCRSCTLR<a>.GROUP == `0b0010` and TRCRSCTLR<a>.SEQUENCER != `0b0000`.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSEQRSTEVR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0x118`	TRCSEQRSTEVR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQSTR, Trace Sequencer State Register

The TRCSEQSTR characteristics are:

Purpose

Use this to set, or read, the Sequencer state.

Configuration

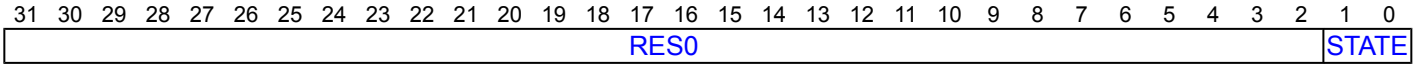
External register TRCSEQSTR bits [31:0] are architecturally mapped to AArch64 System register TRCSEQSTR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQSTR are RES0.

Attributes

TRCSEQSTR is a 32-bit register.

Field descriptions



Bits [31:2]

Reserved, RES0.

STATE, bits [1:0]

Set or returns the state of the Sequencer.

STATE	Meaning
0b00	State 0.
0b01	State 1.
0b10	State 2.
0b11	State 3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQSTR

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.SEQUENCER != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCSEQSTR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x11C	TRCSEQSTR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSCCR<n>, Trace Single-shot Comparator Control Register <n>, n = 0 - 7

The TRCSSCCR<n> characteristics are:

Purpose

Controls the corresponding Single-shot Comparator Control resource.

Configuration

External register TRCSSCCR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCSSCCR<n>[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR4.NUMSSCC) > n. Otherwise, direct accesses to TRCSSCCR<n> are RES0.

Attributes

TRCSSCCR<n> is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
RES0							RST	ARC[7]	ARC[6]	ARC[5]	ARC[4]	ARC[3]	ARC[2]	ARC[1]	ARC[0]	SAC[15]	SAC[14]	SAC[13]	SAC[12]	SAC[11]	SAC[10]

Bits [31:25]

Reserved, RES0.

RST, bit [24]

Selects the Single-shot Comparator Control mode.

RST	Meaning
0b0	The Single-shot Comparator Control is in single-shot mode.
0b1	The Single-shot Comparator Control is in multi-shot mode.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

ARC[<m>], bit [m+16], for m = 7 to 0

Selects one or more Address Range Comparators for Single-shot control.

ARC[<m>]	Meaning
0b0	The Address Range Comparator <m>, is not selected for Single-shot control.
0b1	The Address Range Comparator <m>, is selected for Single-shot control.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is **RES0**.

- Otherwise, access to this field is **RW**.

SAC[<m>], bit [m], for m = 15 to 0

Selects one or more Single Address Comparators for Single-shot control.

SAC[<m>]	Meaning
0b0	The Single Address Comparator <m>, is not selected for Single-shot control.
0b1	The Single Address Comparator <m>, is selected for Single-shot control.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS) * 2, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCSSCCR<n>

Must be programmed if any TRCRSCTLR<a>.GROUP == 0b0011 and TRCRSCTLR<a>.SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSSCCR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x280 + (4 * n)	TRCSSCCR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSCSR<n>, Trace Single-shot Comparator Control Status Register <n>, n = 0 - 7

The TRCSSCSR<n> characteristics are:

Purpose

Returns the status of the corresponding Single-shot Comparator Control.

Configuration

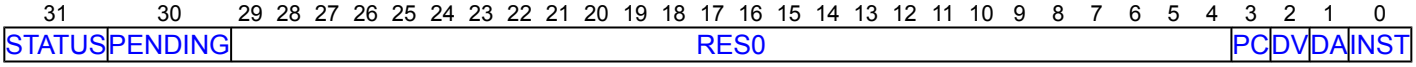
External register TRCSSCSR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCSSCSR<n>[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR4.NUMSSCC) > n. Otherwise, direct accesses to TRCSSCSR<n> are RES0.

Attributes

TRCSSCSR<n> is a 32-bit register.

Field descriptions



STATUS, bit [31]

Single-shot Comparator Control status. Indicates if any of the comparators selected by this Single-shot Comparator control have matched. The selected comparators are defined by TRCSSCCR<n>.ARC, TRCSSCCR<n>.SAC, and TRCSSPCICR<n>.PC.

STATUS	Meaning
0b0	No match has occurred. When the first match occurs, this field takes a value of 1. It remains at 1 until explicitly modified by a write to this register.
0b1	One or more matches has occurred. If TRCSSCCR<n>.RST == 0 then: • There is only one match and no more matches are possible. • Software must reset this field to 0 to re-enable the Single-shot Comparator Control.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

PENDING, bit [30]

Single-shot pending status. The Single-shot Comparator Control fired while the resources were in the Paused state.

PENDING	Meaning
0b0	No match has occurred.
0b1	One or more matches has occurred.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [29:4]

Reserved, RES0.

PC, bit [3]

PE Comparator Input support. Indicates if the Single-shot Comparator Control supports PE Comparator Inputs.

PC	Meaning
0b0	This Single-shot Comparator Control does not support PE Comparator Inputs. Selecting any PE Comparator Inputs using the associated TRCSSPCICR<n> results in CONSTRAINED UNPREDICTABLE behavior of the Single-shot Comparator Control resource. The Single-shot Comparator Control might match unexpectedly or might not match.
0b1	This Single-shot Comparator Control supports PE Comparator Inputs.

Access to this field is **RO**.

DV, bit [2]

Data value comparator support. Data value comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

DV	Meaning
0b0	This Single-shot Comparator Control does not support data value comparisons.
0b1	This Single-shot Comparator Control supports data value comparisons.

This field reads as 0.

Access to this field is **RO**.

DA, bit [1]

Data Address Comparator support. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

DA	Meaning
0b0	This Single-shot Comparator Control does not support data address comparisons.
0b1	This Single-shot Comparator Control supports data address comparisons.

This field reads as 0.

Access to this field is **RO**.

INST, bit [0]

Instruction Address Comparator support. Indicates if the Single-shot Comparator Control supports instruction address comparisons.

INST	Meaning
0b0	This Single-shot Comparator Control does not support instruction address comparisons.
0b1	This Single-shot Comparator Control supports instruction address comparisons.

This field reads as 1.

Access to this field is **RO**.

Accessing TRCSSCSR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0011 and TRCRSCTLR<a>.SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCSSCSR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x2A0 + (4 * n)	TRCSSCSR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSPCICR<n>, Trace Single-shot Processing Element Comparator Input Control Register <n>, n = 0 - 7

The TRCSSPCICR<n> characteristics are:

Purpose

Returns the status of the corresponding Single-shot Comparator Control.

Configuration

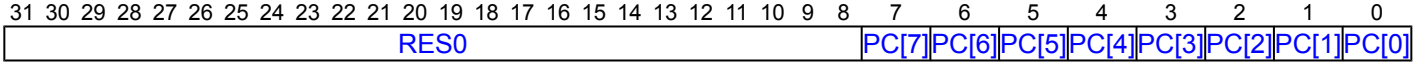
External register TRCSSPCICR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCSSPCICR<n>[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, UInt(TRCIDR4.NUMSSCC) > n, UInt(TRCIDR4.NUMPC) > 0, and TRCSSCSR<n>.PC == 1. Otherwise, direct accesses to TRCSSPCICR<n> are RES0.

Attributes

TRCSSPCICR<n> is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

PC[<m>], bit [m], for m = 7 to 0

Selects one or more PE Comparator Inputs for Single-shot control.

PC[<m>]	Meaning
0b0	The single PE Comparator Input <m>, is not selected as for Single-shot control.
0b1	The single PE Comparator Input <m>, is selected as for Single-shot control.

This bit is RES0 if m >= TRCIDR4.NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSSPCICR<n>

Must be programmed if implemented and any TRCRSCTLR<a>.GROUP == 0b0011 and TRCRSCTLR<a>.SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCSSPCICR<n> can be accessed through the external debug interface:

Component	Offset	Instance

ETE	0x2C0 + (4 * n)	TRCSSPCICR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSTALLCTLR, Trace Stall Control Register

The TRCSTALLCTLR characteristics are:

Purpose

Enables trace unit functionality that prevents trace unit buffer overflows.

Configuration

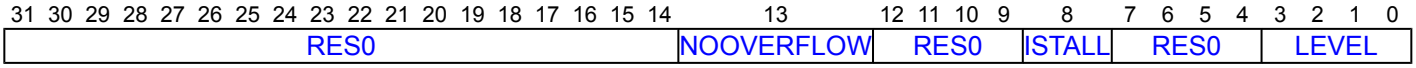
External register TRCSTALLCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCSTALLCTLR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and TRCIDR3.STALLCTL == 1. Otherwise, direct accesses to TRCSTALLCTLR are RES0.

Attributes

TRCSTALLCTLR is a 32-bit register.

Field descriptions



Bits [31:14]

Reserved, RES0.

NOOVERFLOW, bit [13]

When TRCIDR3.NOOVERFLOW == 1:

Trace overflow prevention.

NOOVERFLOW	Meaning
0b0	Trace unit buffer overflow prevention is disabled.
0b1	Trace unit buffer overflow prevention is enabled.

Note

Enabling this feature might cause a significant performance impact.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:9]

Reserved, RES0.

ISTALL, bit [8]

Instruction stall control. Controls if a trace unit can stall the PE when the trace buffer space is less than LEVEL.

ISTALL	Meaning
0b0	The trace unit must not stall the PE.
0b1	The trace unit can stall the PE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

LEVEL, bits [3:0]

Threshold level field. The field can support 16 monotonic levels from 0b0000 to 0b1111.

The value 0b0000 defines the Minimal invasion level. This setting has a greater risk of a trace unit buffer overflow.

The value 0b1111 defines the Maximum invasion level. This setting has a reduced risk of a trace unit buffer overflow.

Note

For some implementations, invasion might occur at the minimal invasion level.

One or more of the least significant bits of LEVEL are permitted to be RES0. Arm recommends that LEVEL[3:2] are fully implemented. Arm strongly recommends that LEVEL[3] is always implemented. If one or more bits are RES0 and are written with a nonzero value, the effective value of LEVEL is rounded down to the nearest power of 2 value which has the RES0 bits as zero. For example, if LEVEL[1:0] are RES0 and a value of 0b1110 is written to LEVEL, the effective value of LEVEL is 0b1100.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSTALLCTLR

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSTALLCTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x02C	TRCSTALLCTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSTATR, Trace Status Register

The TRCSTATR characteristics are:

Purpose

Returns the trace unit status.

Configuration

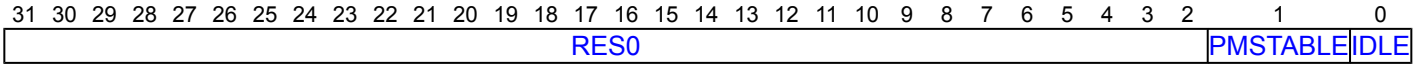
External register TRCSTATR bits [31:0] are architecturally mapped to AArch64 System register TRCSTATR[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCSTATR are RES0.

Attributes

TRCSTATR is a 32-bit register.

Field descriptions



Bits [31:2]

Reserved, RES0.

PMSTABLE, bit [1]

Programmers' model stable.

PMSTABLE	Meaning
0b0	The programmers' model is not stable.
0b1	The programmers' model is stable.

Accessing this field has the following behavior:

- When the trace unit is enabled, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

IDLE, bit [0]

Idle status.

IDLE	Meaning
0b0	The trace unit is not idle.
0b1	The trace unit is idle.

Accessing TRCSTATR

TRCSTATR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x00C	TRCSTATR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSYNCPR, Trace Synchronization Period Register

The TRCSYNCPR characteristics are:

Purpose

Controls how often trace protocol synchronization requests occur.

Configuration

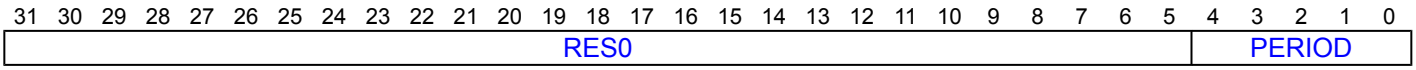
External register TRCSYNCPR bits [31:0] are architecturally mapped to AArch64 System register TRCSYNCPR[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCSYNCPR are RES0.

Attributes

TRCSYNCPR is a 32-bit register.

Field descriptions



Bits [31:5]

Reserved, RES0.

PERIOD, bits [4:0]

Defines the number of bytes of trace between each periodic trace protocol synchronization request.

PERIOD	Meaning
0b00000	Trace protocol synchronization is disabled.
0b01000	Trace protocol synchronization request occurs after 2^8 bytes of trace.
0b01001	Trace protocol synchronization request occurs after 2^9 bytes of trace.
0b01010	Trace protocol synchronization request occurs after 2^{10} bytes of trace.
0b01011	Trace protocol synchronization request occurs after 2^{11} bytes of trace.
0b01100	Trace protocol synchronization request occurs after 2^{12} bytes of trace.
0b01101	Trace protocol synchronization request occurs after 2^{13} bytes of trace.
0b01110	Trace protocol synchronization request occurs after 2^{14} bytes of trace.
0b01111	Trace protocol synchronization request occurs after 2^{15} bytes of trace.
0b10000	Trace protocol synchronization request occurs after 2^{16} bytes of trace.
0b10001	Trace protocol synchronization request occurs after 2^{17} bytes of trace.
0b10010	Trace protocol synchronization request occurs after 2^{18} bytes of trace.
0b10011	Trace protocol synchronization request occurs after 2^{19} bytes of trace.
0b10100	Trace protocol synchronization request occurs after 2^{20} bytes of trace.

Other values are reserved. If a reserved value is programmed into PERIOD, then the behavior of the synchronization period counter is CONSTRAINED UNPREDICTABLE and one of the following behaviors occurs:

- No trace protocol synchronization requests are generated by this counter.
- Trace protocol synchronization requests occur at the specified period.
- Trace protocol synchronization requests occur at some other UNKNOWN period which can vary.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSYNCPR

Must be programmed if TRCIDR3.SYNCPR == 0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSYNCPR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x034	TRCSYNCPR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCTRACEIDR, Trace ID Register

The TRCTRACEIDR characteristics are:

Purpose

Sets the trace ID for instruction trace.

Configuration

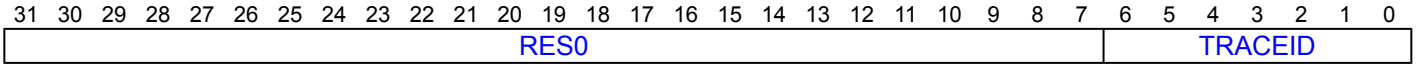
External register TRCTRACEIDR bits [31:0] are architecturally mapped to AArch64 System register TRCTRACEIDR[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCTRACEIDR are RES0.

Attributes

TRCTRACEIDR is a 32-bit register.

Field descriptions



Bits [31:7]

Reserved, RES0.

TRACEID, bits [6:0]

Trace ID field. Sets the trace ID value for instruction trace. The width of the field is indicated by the value of TRCIDR5.TRACEIDSIZE. Unimplemented bits are RES0.

If an implementation supports AMBA ATB, then:

- The width of the field is 7 bits.
- Writing a reserved trace ID value does not affect behavior of the trace unit but it might cause UNPREDICTABLE behavior of the trace capture infrastructure.

See the AMBA ATB Protocol Specification for information about which ATID values are reserved.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCTRACEIDR

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCTRACEIDR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x040	TRCTRACEIDR

Accessible as follows:

- When OSLockStatus(), or !IsTraceCorePowered(), or !AllowExternalTraceAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCTSCTLR, Trace Timestamp Control Register

The TRCTSCTLR characteristics are:

Purpose

Controls the insertion of global timestamps in the trace stream.

Configuration

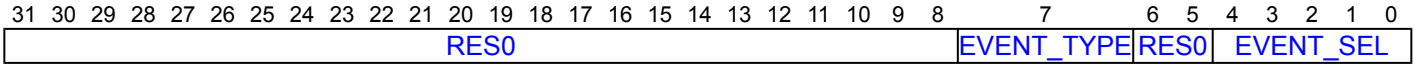
External register TRCTSCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCTSCTLR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and TRCIDR0.TSSIZE != 0b00000. Otherwise, direct accesses to TRCTSCTLR are RES0.

Attributes

TRCTSCTLR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

EVENT_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

EVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCTSCTLR.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCTSCTLR.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCTSCTLR.EVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000:

Defines the selected Resource Selector or pair of Resource Selectors. TRCTSCTLR.EVENT.TYPE controls whether TRCTSCTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCTSCTLR

Must be programmed if TRCCONFIGR.TS == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCTSCTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x030	TRCTSCTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVICTLR, Trace ViewInst Main Control Register

The TRCVICTLR characteristics are:

Purpose

Controls instruction trace filtering.

Configuration

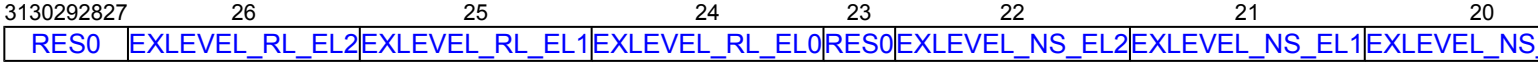
External register TRCVICTLR bits [31:0] are architecturally mapped to AArch64 System register TRCVICTLR[31:0].

This register is present only when FEAT_ETE is implemented and FEAT_TRC_EXT is implemented. Otherwise, direct accesses to TRCVICTLR are RES0.

Attributes

TRCVICTLR is a 32-bit register.

Field descriptions



Bits [31:27]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [26]

When FEAT_RME is implemented:

Filter instruction trace for EL2 in Realm state.

EXLEVEL_RL_EL2	Meaning
0b0	When TRCVICTLR.EXLEVEL_NS_EL2 is 0 the trace unit generates instruction trace for EL2 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL2 is 1 the trace unit does not generate instruction trace for EL2 in Realm state.
0b1	When TRCVICTLR.EXLEVEL_NS_EL2 is 0 the trace unit does not generate instruction trace for EL2 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL2 is 1 the trace unit generates instruction trace for EL2 in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL1, bit [25]

When FEAT_RME is implemented:

Filter instruction trace for EL1 in Realm state.

EXLEVEL_RL_EL1	Meaning
0b0	When TRCVICTLR.EXLEVEL_NS_EL1 is 0 the trace unit generates instruction trace for EL1 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL1 is 1 the trace unit does not generate instruction trace for EL1 in Realm state.
0b1	When TRCVICTLR.EXLEVEL_NS_EL1 is 0 the trace unit does not generate instruction trace for EL1 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL1 is 1 the trace unit generates instruction trace for EL1 in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL0, bit [24]

When FEAT_RME is implemented:

Filter instruction trace for EL0 in Realm state.

EXLEVEL_RL_EL0	Meaning
0b0	When TRCVICTLR.EXLEVEL_NS_EL0 is 0 the trace unit generates instruction trace for EL0 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL0 is 1 the trace unit does not generate instruction trace for EL0 in Realm state.
0b1	When TRCVICTLR.EXLEVEL_NS_EL0 is 0 the trace unit does not generate instruction trace for EL0 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL0 is 1 the trace unit generates instruction trace for EL0 in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]

When Non-secure EL2 is implemented:

Filter instruction trace for EL2 in Non-secure state.

EXLEVEL_NS_EL2	Meaning
0b0	The trace unit generates instruction trace for EL2 in Non-secure state.
0b1	The trace unit does not generate instruction trace for EL2 in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL1, bit [21]
When Non-secure EL1 is implemented:

Filter instruction trace for EL1 in Non-secure state.

EXLEVEL_NS_EL1	Meaning
0b0	The trace unit generates instruction trace for EL1 in Non-secure state.
0b1	The trace unit does not generate instruction trace for EL1 in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL0, bit [20]
When Non-secure EL0 is implemented:

Filter instruction trace for EL0 in Non-secure state.

EXLEVEL_NS_EL0	Meaning
0b0	The trace unit generates instruction trace for EL0 in Non-secure state.
0b1	The trace unit does not generate instruction trace for EL0 in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [19]
When EL3 is implemented:

Filter instruction trace for EL3.

EXLEVEL_S_EL3	Meaning
0b0	The trace unit generates instruction trace for EL3.
0b1	The trace unit does not generate instruction trace for EL3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [18]
When Secure EL2 is implemented:

Filter instruction trace for EL2 in Secure state.

EXLEVEL_S_EL2	Meaning
0b0	The trace unit generates instruction trace for EL2 in Secure state.
0b1	The trace unit does not generate instruction trace for EL2 in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL1, bit [17]
When Secure EL1 is implemented:

Filter instruction trace for EL1 in Secure state.

EXLEVEL_S_EL1	Meaning
0b0	The trace unit generates instruction trace for EL1 in Secure state.
0b1	The trace unit does not generate instruction trace for EL1 in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [16]
When Secure EL0 is implemented:

Filter instruction trace for EL0 in Secure state.

EXLEVEL_S_EL0	Meaning
0b0	The trace unit generates instruction trace for EL0 in Secure state.
0b1	The trace unit does not generate instruction trace for EL0 in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:12]

Reserved, RES0.

TRCERR, bit [11]

When TRCIDR3.TRCERR == 1:

Controls the forced tracing of System Error exceptions.

TRCERR	Meaning
0b0	Forced tracing of System Error exceptions is disabled.
0b1	Forced tracing of System Error exceptions is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCRESET, bit [10]

Controls the forced tracing of PE Resets.

TRCRESET	Meaning
0b0	Forced tracing of PE Resets is disabled.
0b1	Forced tracing of PE Resets is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SSSTATUS, bit [9]

ViewInst start/stop function status.

SSSTATUS	Meaning
0b0	Stopped State. The ViewInst start/stop function is in the stopped state.
0b1	Started State. The ViewInst start/stop function is in the started state.

Before software enables the trace unit, it must write to this field to set the initial state of the ViewInst start/stop function. If the ViewInst start/stop function is not used then set this field to 1. Arm recommends that the value of this field is set before each trace session begins.

If the trace unit becomes disabled while a start point or stop point is still speculative, then the value of TRCVICTLR.SSSTATUS is UNKNOWN and might represent the result of a speculative start point or stop point.

If software which is running on the PE being traced disables the trace unit, either by clearing TRCPRGCTLR.EN or locking the OS Lock, Arm recommends that a DSB and an ISB instruction are executed before disabling the trace unit to prevent any start points or stop points being speculative at the point of disabling the trace unit. This procedure assumes that all start points or stop points occur before the barrier instructions are executed. The procedure does not guarantee that there are no speculative start points or stop points when disabling, although it helps minimize the probability.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is **RES1** if all the following are true:
  - TRCIDR4.NUMACPAIRS == 0b0000.
  - TRCIDR4.NUMPC == 0b0000.

- Otherwise, access to this field is **RW**.

Bit [8]

Reserved, RES0.

EVENT_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

EVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCVICTLR.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCVICTLR.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCVICTLR.EVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

Bits[4:0]

When TRCIDR4.NUMRSPAIR != 0b0000:

EVENT_SEL, bits [4:0] of bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCVICTLR.EVENT.TYPE controls whether TRCVICTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

When TRCIDR4.NUMRSPAIR == 0b0000:

Reserved, bits [4:0] of bits [4:0]

This field is reserved:

- Bits [4:1] are RES0.

- Bit [0] is RES1.

Otherwise:

Reserved, RES0.

Accessing TRCVICTLR

Must be programmed.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCVICTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x080	TRCVICTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVIIECTLR, Trace ViewInst Include/Exclude Control Register

The TRCVIIECTLR characteristics are:

Purpose

Use this to select, or read, the Address Range Comparators for the ViewInst include/exclude function.

Configuration

External register TRCVIIECTLR bits [31:0] are architecturally mapped to AArch64 System register TRCVIIECTLR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR4.NUMACPAIRS) > 0. Otherwise, direct accesses to TRCVIIECTLR are RES0.

Attributes

TRCVIIECTLR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13
RES0								EXCLUDE[7]	EXCLUDE[6]	EXCLUDE[5]	EXCLUDE[4]	EXCLUDE[3]	EXCLUDE[2]	EXCLUDE[1]	EXCLUDE[0]	RE		

Bits [31:24]

Reserved, RES0.

EXCLUDE[<m>], bit [m+16], for m = 7 to 0

Exclude Address Range Comparator <m>. Selects whether Address Range Comparator <m> is in use with the ViewInst exclude function.

EXCLUDE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines, is not selected for the ViewInst exclude function.
0b1	The address range that Address Range Comparator <m> defines, is selected for the ViewInst exclude function.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Bits [15:8]

Reserved, RES0.

INCLUDE[<m>], bit [m], for m = 7 to 0

Include Address Range Comparator <m>.

Selects whether Address Range Comparator <m> is in use with the ViewInst include function.

Selecting no comparators for the ViewInst include function indicates that all instructions are included by default.

The ViewInst exclude function then indicates which ranges are excluded.

INCLUDE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines, is not selected for the ViewInst include function.
0b1	The address range that Address Range Comparator <m> defines, is selected for the ViewInst include function.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCVIIECTLR

Must be programmed if TRCIDR4.NUMACPAIRS > 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVIIECTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x084	TRCVIIECTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVIPCSSCTLR, Trace ViewInst Start/Stop PE Comparator Control Register

The TRCVIPCSSCTLR characteristics are:

Purpose

Use this to select, or read, which PE Comparator Inputs can control the ViewInst start/stop function.

Configuration

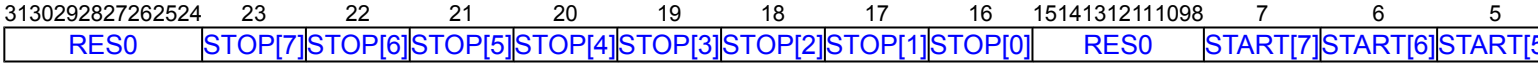
External register TRCVIPCSSCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCVIPCSSCTLR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR4.NUMPC) > 0. Otherwise, direct accesses to TRCVIPCSSCTLR are RES0.

Attributes

TRCVIPCSSCTLR is a 32-bit register.

Field descriptions



Bits [31:24]

Reserved, RES0.

STOP[<m>], bit [m+16], for m = 7 to 0

Selects whether PE Comparator Input <m> is in use with the ViewInst start/stop function for the purpose of stopping trace.

STOP[<m>]	Meaning
0b0	The PE Comparator Input <m> is not selected as a stop resource.
0b1	The PE Comparator Input <m> is selected as a stop resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMPC), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Bits [15:8]

Reserved, RES0.

START[<m>], bit [m], for m = 7 to 0

Selects whether PE Comparator Input <m> is in use with the ViewInst start/stop function for the purpose of starting trace.

START[<m>]	Meaning
0b0	The PE Comparator Input <m> is not selected as a start resource.
0b1	The PE Comparator Input <m> is selected as a start resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMPC), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCVIPCSSCTLR

Must be programmed if TRCIDR4.NUMPC != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVIPCSSCTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x08C	TRCVIPCSSCTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVISSCTLR, Trace ViewInst Start/Stop Control Register

The TRCVISSCTLR characteristics are:

Purpose

Use this to select, or read, the Single Address Comparators for the ViewInst start/stop function.

Configuration

External register TRCVISSCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCVISSCTLR[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR4.NUMACPAIRS) > 0. Otherwise, direct accesses to TRCVISSCTLR are RES0.

Attributes

TRCVISSCTLR is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18
STOP[15]	STOP[14]	STOP[13]	STOP[12]	STOP[11]	STOP[10]	STOP[9]	STOP[8]	STOP[7]	STOP[6]	STOP[5]	STOP[4]	STOP[3]	STOP[2

STOP[<m>], bit [m+16], for m = 15 to 0

Selects whether Single Address Comparator <m> is used with the ViewInst start/stop function for the purpose of stopping trace.

STOP[<m>]	Meaning
0b0	The Single Address Comparator <m> is not selected as a stop resource.
0b1	The Single Address Comparator <m> is selected as a stop resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS) * 2, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

START[<m>], bit [m], for m = 15 to 0

Selects whether Single Address Comparator <m> is used with the ViewInst start/stop function for the purpose of starting trace.

START[<m>]	Meaning
0b0	The Single Address Comparator <m> is not selected as a start resource.
0b1	The Single Address Comparator <m> is selected as a start resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS) * 2, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Accessing TRCVISSCTLR

Must be programmed if TRCIDR4.NUMACPAIRS > 0b0000.

For any 2 comparators selected for the ViewInst start/stop function, the comparator containing the lower address must be a lower numbered comparator.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVISSCTLR can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x088	TRCVISSCTLR

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCCTLR0, Trace Virtual Context Identifier Comparator Control Register 0

The TRCVMIDCCTLR0 characteristics are:

Purpose

Virtual Context Identifier Comparator mask values for the TRCVMIDCVR<n> registers, where n=0-3.

Configuration

External register TRCVMIDCCTLR0 bits [31:0] are architecturally mapped to AArch64 System register TRCVMIDCCTLR0[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, UInt(TRCIDR4.NUMVMIDC) > 0x0, and UInt(TRCIDR2.VMIDSIZE) > 0. Otherwise, direct accesses to TRCVMIDCCTLR0 are RES0.

Attributes

TRCVMIDCCTLR0 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20
COMP3[7]	COMP3[6]	COMP3[5]	COMP3[4]	COMP3[3]	COMP3[2]	COMP3[1]	COMP3[0]	COMP2[7]	COMP2[6]	COMP2[5]	COMP2[4]	C

COMP3[<m>], bit [m+24], for m = 7 to 0
When UInt(TRCIDR4.NUMVMIDC) > 3:

TRCVMIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR3. Each bit in this field corresponds to a byte in TRCVMIDCVR3.

COMP3[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP2[<m>], bit [m+16], for m = 7 to 0
When UInt(TRCIDR4.NUMVMIDC) > 2:

TRCVMIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR2. Each bit in this field corresponds to a byte in TRCVMIDCVR2.

COMP2[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP1[<m>], bit [m+8], for m = 7 to 0
When UInt(TRCIDR4.NUMVMIDC) > 1:

TRCVMIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR1. Each bit in this field corresponds to a byte in TRCVMIDCVR1.

COMP1[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP0[<m>], bit [m], for m = 7 to 0
When UInt(TRCIDR4.NUMVMIDC) > 0:

TRCVMIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR0. Each bit in this field corresponds to a byte in TRCVMIDCVR0.

COMP0[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.

- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Accessing TRCVMIDCCTLR0

If software uses the TRCVMIDCVR<n> registers, where n=0-3, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCVMIDCVR<n> to `0x00`.

If any bit is 1 and the relevant byte in TRCVMIDCVR<n> is not `0x00`, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVMIDCCTLR0 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0x688`	TRCVMIDCCTLR0

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCCTLR1, Trace Virtual Context Identifier Comparator Control Register 1

The TRCVMIDCCTLR1 characteristics are:

Purpose

Virtual Context Identifier Comparator mask values for the TRCVMIDCVR<n> registers, where n=4-7.

Configuration

External register TRCVMIDCCTLR1 bits [31:0] are architecturally mapped to AArch64 System register TRCVMIDCCTLR1[31:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, UInt(TRCIDR4.NUMVMIDC) > 0x4, and UInt(TRCIDR2.VMIDSIZE) > 0. Otherwise, direct accesses to TRCVMIDCCTLR1 are RES0.

Attributes

TRCVMIDCCTLR1 is a 32-bit register.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20
COMP7[7]	COMP7[6]	COMP7[5]	COMP7[4]	COMP7[3]	COMP7[2]	COMP7[1]	COMP7[0]	COMP6[7]	COMP6[6]	COMP6[5]	COMP6[4]	C

COMP7[<m>], bit [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 7:

TRCVMIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR7. Each bit in this field corresponds to a byte in TRCVMIDCVR7.

COMP7[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP6[<m>], bit [m+16], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 6:

TRCVMIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR6. Each bit in this field corresponds to a byte in TRCVMIDCVR6.

COMP6[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR6[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR6[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP5[<m>], bit [m+8], for m = 7 to 0
When UInt(TRCIDR4.NUMVMIDC) > 5:

TRCVMIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR5. Each bit in this field corresponds to a byte in TRCVMIDCVR5.

COMP5[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR5[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR5[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

COMP4[<m>], bit [m], for m = 7 to 0
When UInt(TRCIDR4.NUMVMIDC) > 4:

TRCVMIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR4. Each bit in this field corresponds to a byte in TRCVMIDCVR4.

COMP4[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR4[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR4[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is **RES0**.

- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Accessing TRCVMIDCCTLR1

If software uses the TRCVMIDCVR<n> registers, where n=4-7, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCVMIDCVR<n> to `0x00`.

If any bit is 1 and the relevant byte in TRCVMIDCVR<n> is not `0x00`, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVMIDCCTLR1 can be accessed through the external debug interface:

Component	Offset	Instance
ETE	`0x68C`	TRCVMIDCCTLR1

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCVR<n>, Trace Virtual Context Identifier Comparator Value Register <n>, n = 0 - 7

The TRCVMIDCVR<n> characteristics are:

Purpose

Contains the Virtual Context Identifier Comparator value.

Configuration

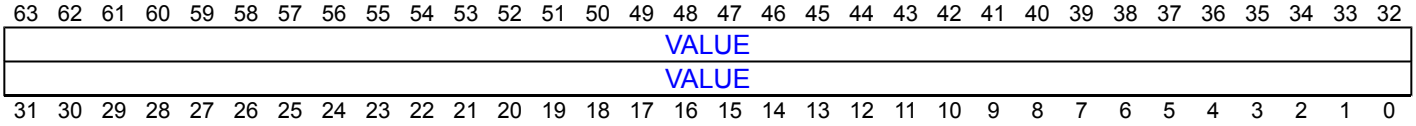
External register TRCVMIDCVR<n> bits [63:0] are architecturally mapped to AArch64 System register TRCVMIDCVR<n>[63:0].

This register is present only when FEAT_ETE is implemented, FEAT_TRC_EXT is implemented, and UInt(TRCIDR4.NUMVMIDC) > n. Otherwise, direct accesses to TRCVMIDCVR<n> are RES0.

Attributes

TRCVMIDCVR<n> is a 64-bit register.

Field descriptions



VALUE, bits [63:0]

Virtual context identifier value. The width of this field is indicated by TRCIDR2.VMIDSIZE. Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Virtual context identifier is zero until the PE updates the Virtual context identifier .

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVMIDCVR<n>

Must be programmed if any of the following are true:

- TRCRSCTLR<a>.GROUP == 0b0111 and TRCRSCTLR<a>.VMID[n] == 1.
- TRCACATR<a>.CONTEXTTYPE == 0b10 or 0b11 and TRCACATR<a>.CONTEXT == n.

TRCVMIDCVR<n> can be accessed through the external debug interface:

Component	Offset	Instance
ETE	0x640 + (8 * n)	TRCVMIDCVR<n>

Accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess(addrdesc), or !IsTraceCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMU

The PMU characteristics are:

Attributes

PMU is a block of size: 4096 bytes

Contents

Offset	Name	Accessor condition	Register condition	Most permissive access
0x000 + (8 * n) for n in 30:0	PMEVCNTR<n>_EL0	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0x000 + (8 * n) for n in 30:0	PMEVCNTR<n>_EL0	When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p5 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0x000 + (8 * n) for n in 30:0	PMEVCNTR<n>_EL0	When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p5 is not implemented	When FEAT_PMUv3_EXT is implemented	RW
0x0F8	PMCCNTR_EL0	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0x0F8	PMCCNTR_EL0	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0x0FC	PMCCNTR_EL0[63:32]	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0x100	PMICNTR_EL0	When FEAT_PMUv3_ICNTR is implemented	When FEAT_PMUv3_ICNTR is implemented	RW
0x200	PMPCSR	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT is implemented and FEAT_PCSRv8p2 is implemented	RO
0x200	PMPCSR	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	When FEAT_PMUv3_EXT is implemented and FEAT_PCSRv8p2 is implemented	RO
0x204	PMPCSR[63:32]	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	When FEAT_PMUv3_EXT is implemented and FEAT_PCSRv8p2 is implemented	RO
0x208	PMVCIDSR	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT64 is implemented and	RO

			FEAT_PCSRv8p2 is implemented	
0x208	PMCID1SR	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	RO
0x20C	PMVIDSR	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	When FEAT_PMUv3_EXT32 is implemented, FEAT_PCSRv8p2 is implemented, and EL2 is implemented	RO
0x220	PMPCSR	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT is implemented and FEAT_PCSRv8p2 is implemented	RO
0x220	PMPCSR	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	When FEAT_PMUv3_EXT is implemented and FEAT_PCSRv8p2 is implemented	RO
0x224	PMPCSR[63:32]	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	When FEAT_PMUv3_EXT is implemented and FEAT_PCSRv8p2 is implemented	RO
0x228	PMCCIDSR	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT64 is implemented and FEAT_PCSRv8p2 is implemented	RO
0x228	PMCID1SR	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	RO
0x22C	PMCID2SR	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	When FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented	RO
0x400 + (8 * n) for n in 30:0	PMEVTYPER<n>_EL0[63:0]	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0x400 + (4 * n) for n in 30:0	PMEVTYPER<n>_EL0[31:0]	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0x47C	PMCCFILTR_EL0[31:0]	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0x480	PMICFILTR_EL0[31:0]	When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3_ICNTR is implemented	When FEAT_PMUv3_ICNTR is implemented	RW
0x4F8	PMCCFILTR_EL0	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT is implemented	RW

0x500	PMICFILTR_EL0	When FEAT_PMUv3_EXT64 is implemented and FEAT_PMUv3_ICNTR is implemented	When FEAT_PMUv3_ICNTR is implemented	RW
0x600 + (8 * n) for n in 30:0	PMEVCNTSVR<n>_EL1	When FEAT_PMUv3_SS is implemented	When FEAT_PMUv3_SS is implemented	RO
0x6F8	PMCCNTSVR_EL1	When FEAT_PMUv3_SS is implemented	When FEAT_PMUv3_SS is implemented	RO
0x700	PMICNTSVR_EL1	When FEAT_PMUv3_SS is implemented and FEAT_PMUv3_ICNTR is implemented	When FEAT_PMUv3_ICNTR is implemented and FEAT_PMUv3_SS is implemented	RO
0x800 + (4 * n) for n in 63:0	PMEVFILT2R<n>[31:0]	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMEVFILT2R<n>	RW
0x800 + (8 * n) for n in 63:0	PMEVFILT2R<n>[63:0]	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMEVFILT2R<n>	RW
0xA00 + (4 * n) for n in 30:0	PMEVTYPER<n>_EL0[63:32]	When FEAT_PMUv3_EXT32 is implemented and (FEAT_PMUv3_TH is implemented, or FEAT_PMUv3p8 is implemented, or FEAT_PMUv3_SME is implemented)	When FEAT_PMUv3_EXT is implemented	RW
0xA7C	PMCCFILTR_EL0[63:32]	When FEAT_PMUv3_EXT32 is implemented and (FEAT_PMUv3_TH is implemented, or FEAT_PMUv3p8 is implemented, or FEAT_PMUv3_SME is implemented)	When FEAT_PMUv3_EXT is implemented	RW
0xA80	PMICFILTR_EL0[63:32]	When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3_ICNTR is implemented	When FEAT_PMUv3_ICNTR is implemented	RW
0xC00	PMCNTENSET_EL0	When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0xC00	PMCNTENSET_EL0	When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented	When FEAT_PMUv3_EXT is implemented	RW

0xC10	PMCNTEN	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT64 is implemented	RW
0xC20	PMCNTENCLR_EL0	When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0xC20	PMCNTENCLR_EL0	When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented	When FEAT_PMUv3_EXT is implemented	RW
0xC40	PMINTENSET_EL1	When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0xC40	PMINTENSET_EL1	When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented	When FEAT_PMUv3_EXT is implemented	RW
0xC50	PMINTEN	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT64 is implemented	RW
0xC60	PMINTENCLR_EL1	When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0xC60	PMINTENCLR_EL1	When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented	When FEAT_PMUv3_EXT is implemented	RW
0xC80	PMOVSCLR_EL0	When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0xC80	PMOVSCLR_EL0	When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented	When FEAT_PMUv3_EXT is implemented	RW

0xC90	PMOVS	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT64 is implemented	RW
0xCA0	PMSWINC_EL0	When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented	When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3p9 is not implemented, and an implementation implements PMSWINC_EL0	WO
0xCA0	PMZR_EL0	When FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p9 is implemented	When FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p9 is implemented	WO
0xCC0	PMOVSSET_EL0	When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0xCC0	PMOVSSET_EL0	When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented	When FEAT_PMUv3_EXT is implemented	RW
0xCE0	PMCGCR0	When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3_ICNTR is implemented	When FEAT_PMUv3_ICNTR is implemented	RO
0xCE0	PMCGCR0	When FEAT_PMUv3_EXT64 is implemented and FEAT_PMUv3_ICNTR is implemented	When FEAT_PMUv3_ICNTR is implemented	RO
0xE00	PMCFGR	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT is implemented	RO
0xE00	PMCFGR	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT is implemented	RO
0xE04	PMCR_EL0	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0xE08	PMIIDR	When FEAT_PMUv3_EXT is implemented	When (FEAT_PMUv3_EXT32 is implemented and an implementation implements PMIIDR) or FEAT_PMUv3_EXT64 is implemented	RO
0xE10	PMCR_EL0	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT is implemented	RW
0xE20	PMCEID0	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT32 is implemented	RO

0xE24	PMCEID1	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT32 is implemented	RO
0xE28	PMCEID2	When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p1 is implemented	When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p1 is implemented	RO
0xE2C	PMCEID3	When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p1 is implemented	When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p1 is implemented	RO
0xE30	PMSSCR_EL1	When FEAT_PMUv3_SS is implemented	When FEAT_PMUv3_SS is implemented	RW
0xE40	PMMIR	When FEAT_PMUv3p4 is implemented and (FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented)	When FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p4 is implemented	RO
0xE40	PMMIR	When FEAT_PMUv3p4 is implemented, FEAT_PMUv3_EXT32 is implemented, and FEAT_PMUv3p9 is not implemented	When FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p4 is implemented	RO
0xE50	PMPCSCTL	When FEAT_PCSRv8p9 is implemented	When FEAT_PCSRv8p9 is implemented	RW
0xE58	PMCCR	When FEAT_PMUv3_EXTPMN is implemented	When FEAT_PMUv3_EXTPMN is implemented	RW
0xF00	PMITCTRL	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMITCTRL	RW
0xFA8	PMDEVAFF	When FEAT_PMUv3_EXT64 is implemented	When FEAT_PMUv3_EXT64 is implemented	RO
0xFA8	PMDEVAFF0	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT32 is implemented	RO
0xFAC	PMDEVAFF1	When FEAT_PMUv3_EXT32 is implemented	When FEAT_PMUv3_EXT32 is implemented	RO
0xFB0	PMLAR	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented	WO
0xFB4	PMLSR	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented	RO
0xFB8	PMAUTHSTATUS	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented	RO
0xFBC	PMDEVARCH	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented	RO

0xFC8	PMDEVID	When FEAT_PMUv3_EXT is implemented and (v8Ap2 or FEAT_PCSRv8p2 is implemented)	When (v8Ap2 or FEAT_PCSRv8p2 is implemented) and FEAT_PMUv3_EXT is implemented	RO
0xFCC	PMDEVTYPE	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMDEVTYPE	RO
0xFD0	PMPIDR4	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMPIDR4	RO
0xFE0	PMPIDR0	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMPIDR0	RO
0xFE4	PMPIDR1	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMPIDR1	RO
0xFE8	PMPIDR2	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMPIDR2	RO
0xFEC	PMPIDR3	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMPIDR3	RO
0xFF0	PMCIDR0	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMCIDR0	RO
0xFF4	PMCIDR1	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMCIDR1	RO
0xFF8	PMCIDR2	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMCIDR2	RO
0xFFC	PMCIDR3	When FEAT_PMUv3_EXT is implemented	When FEAT_PMUv3_EXT is implemented and an implementation implements PMCIDR3	RO

Direct accesses to other offsets in this block are RES0.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMAUTHSTATUS, Performance Monitors Authentication Status register

The PMAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for Performance Monitors.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMAUTHSTATUS are RES0.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

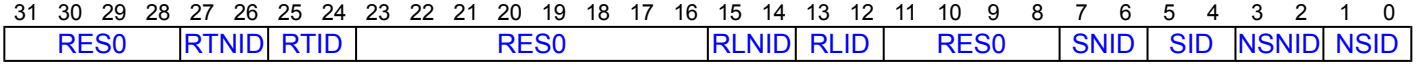
This register is OPTIONAL, and is required for CoreSight compliance. Arm recommends that this register is implemented.

Attributes

PMAUTHSTATUS is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTNID.

RTID, bits [25:24]

Root invasive debug.

RTID	Meaning
0b00	Not implemented.

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLNID.

RLID, bits [13:12]

Realm invasive debug.

RLID	Meaning
0b00	Not implemented.

Bits [11:8]

Reserved, RES0.

SNID, bits [7:6]

Holds the same value as DBGAUTHSTATUS_EL1.SNID.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SID, bits [5:4]

Secure invasive debug.

SID	Meaning
0b00	Not implemented.

All other values are reserved.

Access to this field is **RO**.

NSNID, bits [3:2]

Holds the same value as DBGAUTHSTATUS_EL1.NSNID.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

NSID, bits [1:0]

Non-secure invasive debug.

NSID	Meaning
0b00	Not implemented.

All other values are reserved.

Access to this field is **RO**.

Accessing PMAUTHSTATUS

Accesses to this register use the following encodings:

Accessible at offset 0xFB8 from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCFILTR_EL0, Performance Monitors Cycle Counter Filter Register

The PMCCFILTR_EL0 characteristics are:

Purpose

Determines the modes in which the Cycle Counter, PMCCNTR_EL0, increments.

Configuration

External register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMCCFILTR_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented.

External register PMCCFILTR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCCFILTR_EL0[63:0] when FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_TH is implemented, or FEAT_PMUv3p8 is implemented.

External register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCCFILTR[31:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMCCFILTR_EL0 are RES0.

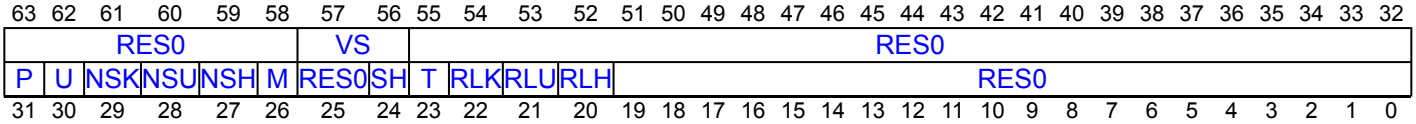
PMCCFILTR_EL0 is in the Core power domain.

Attributes

PMCCFILTR_EL0 is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:58]

Reserved, RES0.

VS, bits [57:56]

When FEAT_PMUv3_SME is implemented:

SVE mode filtering. Controls counting cycles in Streaming and Non-streaming SVE modes.

VS	Meaning
0b00	This mechanism has no effect on the filtering of cycles.
0b01	The PE does not count cycles in Streaming SVE mode.
0b10	The PE does not count cycles in Non-streaming SVE mode.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [55:32]

Reserved, RES0.

P, bit [31]

EL1 filtering. Controls counting cycles in EL1.

P	Meaning
0b0	This mechanism has no effect on filtering of cycles.
0b1	The PE does not count cycles in EL1.

If Secure and Non-secure states are implemented, then counting cycles in Non-secure EL1 is further controlled by PMCCFILTR_EL0.NSK.

If FEAT_RME is implemented, then counting cycles in Realm EL1 is further controlled by PMCCFILTR_EL0.RLK.

If EL3 is implemented, then counting cycles in EL3 is further controlled by PMCCFILTR_EL0.M.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

EL0 filtering. Controls counting cycles in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of cycles.
0b1	The PE does not count cycles in EL0.

If Secure and Non-secure states are implemented, then counting cycles in Non-secure EL0 is further controlled by PMCCFILTR_EL0.NSU.

If FEAT_RME is implemented, then counting cycles in Realm EL0 is further controlled by PMCCFILTR_EL0.RLU.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting cycles in Non-secure EL1. If PMCCFILTR_EL0.NSK is not equal to PMCCFILTR_EL0.P, then the PE does not count cycles in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of cycles in Non-secure EL1.

NSK	Meaning
0b0	When PMCCFILTR_EL0.P == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.P == 1, the PE does not count cycles in Non-secure EL1.
0b1	When PMCCFILTR_EL0.P == 0, the PE does not count cycles in Non-secure EL1. When PMCCFILTR_EL0.P == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]
When EL3 is implemented:

Non-secure EL0 filtering. Controls counting cycles in Non-secure EL0. If PMCCFILTR_EL0.NSU is not equal to PMCCFILTR_EL0.U, then the PE does not count cycles in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of cycles in Non-secure EL0.

NSU	Meaning
0b0	When PMCCFILTR_EL0.U == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.U == 1, the PE does not count cycles in Non-secure EL0.
0b1	When PMCCFILTR_EL0.U == 0, the PE does not count cycles in Non-secure EL0. When PMCCFILTR_EL0.U == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]
When EL2 is implemented:

EL2 filtering. Controls counting cycles in EL2.

NSH	Meaning
0b0	The PE does not count cycles in EL2.
0b1	This mechanism has no effect on filtering of cycles.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting cycles in Secure EL2 is further controlled by PMCCFILTR_EL0.SH.

If FEAT_RME is implemented, then counting cycles in Realm EL2 is further controlled by PMCCFILTR_EL0.RLH.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]

When EL3 is implemented and FEAT_AA64 is implemented:

EL3 filtering. Controls counting cycles in EL3. If PMCCFILTR_EL0.M is not equal to PMCCFILTR_EL0.P, then the PE does not count cycles in EL3. Otherwise, this mechanism has no effect on filtering of cycles in EL3.

M	Meaning
0b0	When PMCCFILTR_EL0.P == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.P == 1, the PE does not count cycles in EL3.
0b1	When PMCCFILTR_EL0.P == 0, the PE does not count cycles in EL3. When PMCCFILTR_EL0.P == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [25]

Reserved, RES0.

SH, bit [24]

When EL3 is implemented and FEAT_SEL2 is implemented:

Secure EL2 filtering. Controls counting cycles in Secure EL2. If PMCCFILTR_EL0.SH is equal to PMCCFILTR_EL0.NSH, then the PE does not count cycles in Secure EL2. Otherwise, this mechanism has no effect on filtering of cycles in Secure EL2.

SH	Meaning
0b0	When PMCCFILTR_EL0.NSH == 0, the PE does not count cycles in Secure EL2. When PMCCFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of cycles.
0b1	When PMCCFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.NSH == 1, the PE does not count cycles in Secure EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When Secure EL2 is not implemented, access to this field is **RES0**.

Otherwise:

Reserved, RES0.

T, bit [23]

When FEAT_TME is implemented:

Non-Transactional state filtering field. Controls counting of cycles in Non-transactional state.

T	Meaning
0b0	This field has no effect on the filtering of cycles.
0b1	Do not count Attributable cycles in Non-transactional state.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLK, bit [22]

When FEAT_RME is implemented:

Realm EL1 filtering. Controls counting cycles in Realm EL1. If PMCCFILTR_EL0.RLK is not equal to PMCCFILTR_EL0.P, then the PE does not count cycles in Realm EL1. Otherwise, this mechanism has no effect on filtering of cycles in Realm EL1.

RLK	Meaning
0b0	When PMCCFILTR_EL0.P == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.P == 1, the PE does not count cycles in Realm EL1.
0b1	When PMCCFILTR_EL0.P == 0, the PE does not count cycles in Realm EL1. When PMCCFILTR_EL0.P == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting cycles in Realm EL0. If PMCCFILTR_EL0.RLU is not equal to PMCCFILTR_EL0.U, then the PE does not count cycles in Realm EL0. Otherwise, this mechanism has no effect on filtering of cycles in Realm EL0.

RLU	Meaning
0b0	When PMCCFILTR_EL0.U == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.U == 1, the PE does not count cycles in Realm EL0.
0b1	When PMCCFILTR_EL0.U == 0, the PE does not count cycles in Realm EL0. When PMCCFILTR_EL0.U == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]

When FEAT_RME is implemented:

Realm EL2 filtering. Controls counting cycles in Realm EL2. If PMCCFILTR_EL0.RLH is equal to PMCCFILTR_EL0.NSH, then the PE does not count cycles in Realm EL2. Otherwise, this mechanism has no effect on filtering of cycles in Realm EL2.

RLH	Meaning
0b0	When PMCCFILTR_EL0.NSH == 0, the PE does not count cycles in Realm EL2. When PMCCFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of cycles.
0b1	When PMCCFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.NSH == 1, the PE does not count cycles in Realm EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:0]

Reserved, RES0.

Accessing PMCCFILTR_EL0

If FEAT_PMUv3_EXT32 is implemented, and any of the following apply, then bits [63:32] of this register are accessible at offset 0xA7C:

- FEAT_PMUv3_TH is implemented.
- FEAT_PMUv3p8 is implemented.
- FEAT_PMUv3_SME is implemented.

Otherwise accesses at this offset are IMPLEMENTATION DEFINED.

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT32 is implemented

[31:0] Accessible at offset 0x47C from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT64 is implemented

Accessible at offset `0x4F8` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented and (FEAT_PMUv3_TH is implemented, or FEAT_PMUv3p8 is implemented, or FEAT_PMUv3_SME is implemented)

[63:32] Accessible at offset `0xA7C` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCIDSR, CONTEXTIDR_ELx Sample Register

The PMCCIDSR characteristics are:

Purpose

Contains the sampled value of CONTEXTIDR_EL1 and CONTEXTIDR_EL2, captured on reading PMPCSR.

Configuration

External register PMCCIDSR bits [31:0] are architecturally mapped to External register PMVCIDSR[31:0].

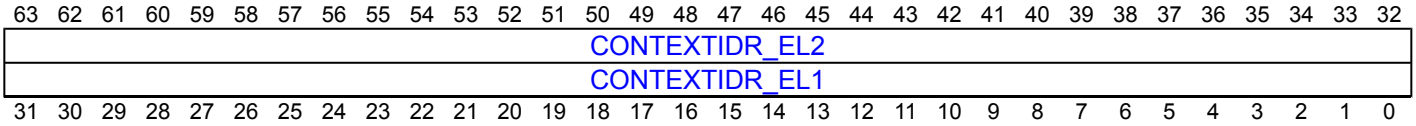
This register is present only when FEAT_PMUv3_EXT64 is implemented and FEAT_PCSRv8p2 is implemented.

Attributes

PMCCIDSR is a 64-bit register.

This register is part of the PMU block.

Field descriptions



CONTEXTIDR_EL2, bits [63:32]

Context ID. The value of CONTEXTIDR_EL2 that is associated with the most recent PMPCSR sample. When the most recent PMPCSR sample is generated:

- If the PE is not executing at EL3, EL2 is using AArch64, and EL2 is enabled in the current Security state, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- Otherwise, this field is set to an UNKNOWN value.

Because the value written to this field is an indirect read of CONTEXTIDR_EL2, it is CONSTRAINED UNPREDICTABLE whether this field is set to the original or new value if PMPCSR samples:

- An instruction that writes to CONTEXTIDR_EL2.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CONTEXTIDR_EL1, bits [31:0]

Context ID. The value of CONTEXTIDR that is associated with the most recent PMPCSR sample. When the most recent PMPCSR sample is generated:

- If EL1 is using AArch64, then the Context ID is sampled from CONTEXTIDR_EL1.
- If EL1 is using AArch32, then the Context ID is sampled from CONTEXTIDR.
- If EL3 is implemented and is using AArch32, then CONTEXTIDR is a banked register and this register samples the current banked copy of CONTEXTIDR for the Security state that is associated with the most recent PMPCSR sample.

Because the value written to this register is an indirect read of CONTEXTIDR, it is CONSTRAINED UNPREDICTABLE whether this register is set to the original or new value if PMPCSR samples:

- An instruction that writes to CONTEXTIDR.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCCIDSR

If FEAT_PCSRv8p2 and FEAT_PMUv3_EXT32 are implemented, then the same content is present in the same locations, and can be accessed using PMCID2SR[31:0] and PMCID1SR[31:0].

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

Accesses to this register use the following encodings:

Accessible at offset `0x228` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCNTR_EL0, Performance Monitors Cycle Counter

The PMCCNTR_EL0 characteristics are:

Purpose

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. For more information, see 'Time as measured by the Performance Monitors cycle counter'.

PMCCFILTR_EL0 determines the modes and states in which the PMCCNTR_EL0 can increment.

Configuration

External register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCCNTR_EL0[63:0].

External register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch32 System register PMCCNTR[63:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMCCNTR_EL0 are RES0.

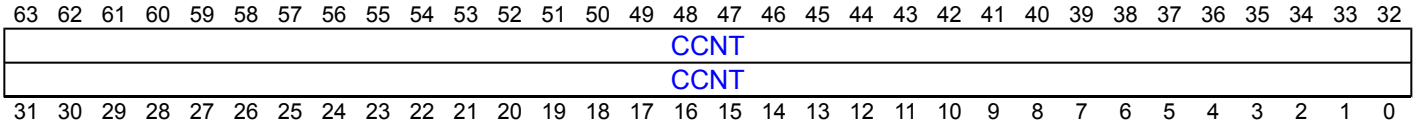
PMCCNTR_EL0 is in the Core power domain.

Attributes

PMCCNTR_EL0 is a 64-bit register.

This register is part of the PMU block.

Field descriptions



CCNT, bits [63:0]

Cycle count. Depending on the values of PMCR_EL0.{LC,D}, the cycle count increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to PMCR_EL0.C sets this field to 0.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMCCNTR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented

[63:0] Accessible at offset `0x0F8` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented

[31:0] Accessible at offset `0x0F8` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented

[63:32] Accessible at offset `0x0FC` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCNTSVR_EL1, Performance Monitors Cycle Count Saved Value Register

The PMCCNTSVR_EL1 characteristics are:

Purpose

Captures the PMU Cycle counter, PMCCNTR_EL0.

Configuration

External register PMCCNTSVR_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMCCNTSVR_EL1[63:0].

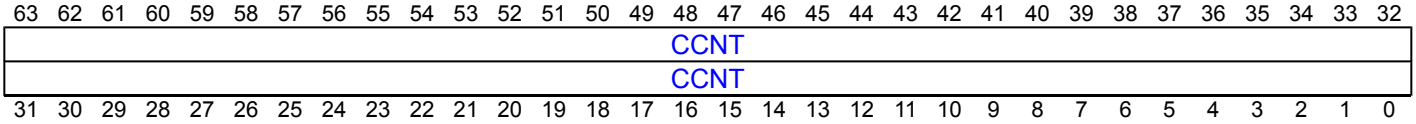
This register is present only when FEAT_PMUv3_SS is implemented. Otherwise, direct accesses to PMCCNTSVR_EL1 are RES0.

Attributes

PMCCNTSVR_EL1 is a 64-bit register.

This register is part of the PMU block.

Field descriptions



CCNT, bits [63:0]

Sampled Cycle Count. The value of PMCCNTR_EL0 at the last successful Capture event.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMCCNTSVR_EL1

Accesses to this register use the following encodings:

Accessible at offset `0x6F8` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMSSAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCR, PMU Configuration Control Register

The PMCCR characteristics are:

Purpose

Contains PMU configuration controls.

Configuration

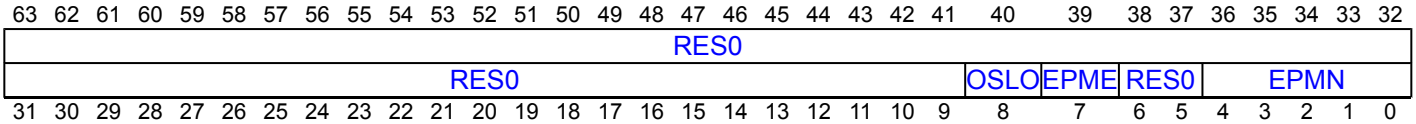
This register is present only when FEAT_PMUv3_EXTPMN is implemented. Otherwise, direct accesses to PMCCR are RES0.

Attributes

PMCCR is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:9]

Reserved, RES0.

OSLO, bit [8]

When FEAT_PMUv3_EXTPMN is implemented:

OS Lock Override.

OSLO	Meaning
0b0	No external access to any Performance Monitor register is affected by this control.
0b1	For the purpose of determining the access permissions of Performance Monitor registers, an external access that is a Most secure access ignores OSLSR_EL1.OSLK.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

EPME, bit [7]

When FEAT_PMUv3_EXTPMN is implemented:

External Enable.

EPME	Meaning
0b0	Affected counters are disabled and do not count.
0b1	Affected counters are enabled by PMCNTENSET_EL0.

The counters affected by this field are the event counters in the third range.

Other event counters, PMCCNTR_EL0, and, if FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0 are not affected by this field.

If the Effective value of PMCCR.EPMN is equal to NUM_PMU_COUNTERS, then this field has no effect.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EPMN, bits [4:0]

When FEAT_PMUv3_EXTPMN is implemented:

Defines the number of event counters PMEVCNTR<n>_EL0 and, if FEAT_PMUv3_SS is implemented, snapshot registers PMEVCNTSVR<n>_EL1, that are reserved for external use.

PMCCR.EPMN divides the event counters into event counters that are accessible from self-hosted software, and which might be further divided into first and second ranges by MDCR_EL2.HPMN, and a third range that is inaccessible from self-hosted software.

If PMCCR.EPMN is not 0 and is less than the number of PMU event counters implemented by the PE, NUM_PMU_COUNTERS, then event counters [0..(PMCCR.EPMN-1)] are in the first and second ranges, and event counters [PMCCR.EPMN..(NUM_PMU_COUNTERS-1)] are in the third range.

If PMCCR.EPMN is equal to NUM_PMU_COUNTERS, or FEAT_PMUv3_EXTPMN is not implemented, then all of the following apply:

- All counters are in the first and second ranges.
- No counters are in the third range.

If PMCCR.EPMN is zero, then all of the following apply:

- No counters are in the first and second ranges.
- All counters are in the third range.

All the following apply for an event counter PMEVCNTR<n>_EL0 in the third range:

- The counter is accessible to a Most secure access of PMEVCNTR<n>_EL0. That is, an external access which is one of the following:
  - Root access, when FEAT_RME is implemented.
  - Secure access, when FEAT_RME is not implemented and Secure state is implemented.
  - Non-secure access, otherwise.
- The counter is not accessible to other external accesses, or as the System register PMEVCNTR<n>_EL0 at any Exception level.
- The counter overflow flag PMOVSSET_EL0[n] is set on unsigned overflow of PMEVCNTR<n>_EL0[63:0].
- PMCCR.EPME and PMCNTENSET_EL0 enable operation of the event counter.

If FEAT_PMUv3_SS is implemented, then all of the following apply for an event counter snapshot register PMEVCNTSVR<n>_EL1 in the third range:

- The event counter snapshot register is accessible to a Most secure access of PMEVCNTSVR<n>_EL1.
- The event counter snapshot register is not accessible to other external accesses, or as the System register PMEVCNTSVR<n>_EL1 at any Exception level.

For information about counters in the first and second ranges, see the description of MDCR_EL2.HPMN.

Values greater than NUM_PMU_COUNTERS are reserved.

If this field is set to a reserved value, then the following CONSTRAINED UNPREDICTABLE behaviors apply:

- The value returned by an external read of PMCCR.EPMN is UNKNOWN and less than or equal to 31.
- For the purpose of indirect reads of PMCCR.EPMN as a result of the following reads, the Effective value of PMCCR.EPMN is UNKNOWN and less than or equal to 31:
  - Direct reads of PMCR_EL0.N or PMCR.N.
  - Direct reads of MDCR_EL2.HPMN.
  - External reads of PMCFGR.N.
  - If FEAT_PMUv3_ICNTR is implemented, external reads of PMCGCR0.CG0NC.
- For all other purposes the Effective value of PMCCR.EPMN is UNKNOWN and less than or equal to NUM_PMU_COUNTERS.

If FEAT_PMUv3_EXTPMN is not implemented, then the Effective value of PMCCR.EPMN is NUM_PMU_COUNTERS.

The reset behavior of this field is:

- On a Cold reset, this field resets to the expression `NUM_PMU_COUNTERS`.

Otherwise:

Reserved, RES0.

Accessing PMCCR

Accesses to this register use the following encodings:

Accessible at offset `0xE58` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When !IsMostSecureAccess(addrdesc), accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID0, Performance Monitors Common Event Identification register 0

The PMCEID0 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range `0x0000` to `0x001F`.

For more information about the Common events and the use of the PMCEIDn registers, see 'The PMU event number space and common events'.

Use of this register is deprecated.

Note

This view of the register was previously called PMCEID0_EL0.

Configuration

External register PMCEID0 bits [31:0] are architecturally mapped to AArch64 System register PMCEID0_EL0[31:0].

External register PMCEID0 bits [31:0] are architecturally mapped to AArch32 System register PMCEID0[31:0].

This register is present only when FEAT_PMUv3_EXT32 is implemented. Otherwise, direct accesses to PMCEID0 are RES0.

PMCEID0 is in the Core power domain.

Attributes

PMCEID0 is a 32-bit register.

This register is part of the PMU block.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4
ID31	ID30	ID29	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event n.

For each bit:

ID<n>	Meaning
`0b0`	The Common event is not implemented, or not counted.
`0b1`	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID0

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

Accessible at offset `0xE20` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID1, Performance Monitors Common Event Identification register 1

The PMCEID1 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range `0x020` to `0x03F`.

For more information about the Common events and the use of the PMCEIDn registers, see 'The PMU event number space and common events'.

Use of this register is deprecated.

Note

This view of the register was previously called PMCEID1_EL0.

Configuration

External register PMCEID1 bits [31:0] are architecturally mapped to AArch64 System register PMCEID1_EL0[31:0].

External register PMCEID1 bits [31:0] are architecturally mapped to AArch32 System register PMCEID1[31:0].

This register is present only when FEAT_PMUv3_EXT32 is implemented. Otherwise, direct accesses to PMCEID1 are RES0.

PMCEID1 is in the Core power domain.

Attributes

PMCEID1 is a 32-bit register.

This register is part of the PMU block.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4
ID31	ID30	ID29	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event (`0x0020` + n).

For each bit:

ID<n>	Meaning
`0b0`	The Common event is not implemented, or not counted.
`0b1`	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID1

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

Accessible at offset `0xE24` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID2, Performance Monitors Common Event Identification register 2

The PMCEID2 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range `0x4000` to `0x401F`.

For more information about the Common events and the use of the PMCEIDn registers, see 'The PMU event number space and common events'.

Use of this register is deprecated.

Configuration

External register PMCEID2 bits [31:0] are architecturally mapped to AArch64 System register PMCEID0_EL0[63:32].

External register PMCEID2 bits [31:0] are architecturally mapped to AArch32 System register PMCEID2[31:0].

This register is present only when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p1 is implemented. Otherwise, direct accesses to PMCEID2 are RES0.

PMCEID2 is in the Core power domain.

Attributes

PMCEID2 is a 32-bit register.

This register is part of the PMU block.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	
IDhi31	IDhi30	IDhi29	IDhi28	IDhi27	IDhi26	IDhi25	IDhi24	IDhi23	IDhi22	IDhi21	IDhi20	IDhi19	IDhi18	IDhi17	IDhi16	IDhi15	IDhi14	IDhi13	IDh

IDhi<n>, bit [n], for n = 31 to 0

IDhi[n] corresponds to Common event (`0x4000` + n).

For each bit:

IDhi<n>	Meaning
`0b0`	The Common event is not implemented, or not counted.
`0b1`	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID2

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

Accessible at offset `0xE28` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID3, Performance Monitors Common Event Identification register 3

The PMCEID3 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range `0x4020` to `0x403F`.

For more information about the Common events and the use of the PMCEIDn registers, see 'The PMU event number space and common events'.

Use of this register is deprecated.

Configuration

External register PMCEID3 bits [31:0] are architecturally mapped to AArch64 System register PMCEID1_EL0[63:32].

External register PMCEID3 bits [31:0] are architecturally mapped to AArch32 System register PMCEID3[31:0].

This register is present only when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p1 is implemented. Otherwise, direct accesses to PMCEID3 are RES0.

PMCEID3 is in the Core power domain.

Attributes

PMCEID3 is a 32-bit register.

This register is part of the PMU block.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	
IDhi31	IDhi30	IDhi29	IDhi28	IDhi27	IDhi26	IDhi25	IDhi24	IDhi23	IDhi22	IDhi21	IDhi20	IDhi19	IDhi18	IDhi17	IDhi16	IDhi15	IDhi14	IDhi13	IDh

IDhi<n>, bit [n], for n = 31 to 0

IDhi[n] corresponds to Common event (`0x4020` + n).

For each bit:

IDhi<n>	Meaning
`0b0`	The Common event is not implemented, or not counted.
`0b1`	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID3

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

Accessible at offset `0xE2C` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCFGR, Performance Monitors Configuration Register

The PMCFGR characteristics are:

Purpose

Contains PMU-specific configuration data.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMCFGR are RES0.

PMCFGR is in the Core power domain.

Attributes

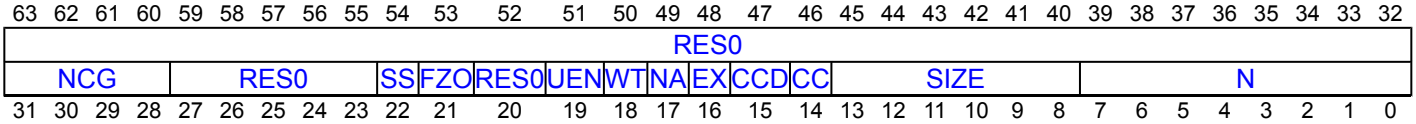
PMCFGR is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented:



Bits [63:32]

Reserved, RES0.

NCG, bits [31:28]

Defines the number of counter groups implemented, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCG	Meaning
0b0000	One counter group implemented.
0b0001	Two counter groups implemented.

All other values are reserved.

FEAT_PMUv3_ICNTR implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

Bits [27:23]

Reserved, RES0.

SS, bit [22]

Snapshot supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SS	Meaning
0b0	Snapshot mechanism not supported. The locations 0x600-0x7FC and 0xE30-0xE3C are IMPLEMENTATION DEFINED.
0b1	Snapshot mechanism supported. If FEAT_PMUv3_SS is implemented, then the following registers are implemented: • PMEVCNTSVR<n>_EL1. • PMCCNTSVR_EL1. • If FEAT_PMUv3_ICNTR is implemented, PMICNTSVR_EL1. • PMSSCR_EL1. Otherwise, locations 0x600-0x7FC and 0xE30-0xE3C contain IMPLEMENTATION DEFINED snapshot registers.

FEAT_PMUv3_SS implements the functionality identified by the value 1.

If FEAT_PMUv3_SS is not implemented, a PMU might include an IMPLEMENTATION DEFINED snapshot mechanism, including one using the IMPLEMENTATION DEFINED registers 0x600-0x7FC and 0xE30-0xE3C.

Access to this field is **RO**.

FZO, bit [21]

Freeze-on-overflow supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FZO	Meaning
0b0	Freeze-on-overflow mechanism is not supported. PMCR_EL0.FZO is RES0.
0b1	Freeze-on-overflow mechanism is supported. PMCR_EL0.FZO is RW.

FEAT_PMUv3p7 implements the functionality added by the value 0b1.

From Armv8.7, if FEAT_PMUv3 is implemented, the only permitted value is 0b1.

Access to this field is **RO**.

Bit [20]

Reserved, RES0.

UEN, bit [19]

User-mode Enable Register supported. PMUSERENR_EL0 is not visible in the external debug interface, so this bit is RAZ.

Reads as 0b0.

Access to this field is **RO**.

WT, bit [18]

This feature is not supported, so this bit is RAZ.

Reads as 0b0.

Access to this field is **RO**.

NA, bit [17]

This feature is not supported, so this bit is RAZ.

Reads as 0b0.

Access to this field is **RO**.

EX, bit [16]

Export supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EX	Meaning
0b0	PMCR_EL0.X is RES0.
0b1	PMCR_EL0.X is read/write.

Access to this field is **RO**.

CCD, bit [15]

Cycle counter has prescale.

This is RES1 if AArch32 is supported, and RAZ otherwise.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCD	Meaning
0b0	PMCR_EL0.D is RES0.
0b1	PMCR_EL0.D is read/write.

Access to this field is **RO**.

CC, bit [14]

Dedicated cycle counter (counter 31) supported.

Reads as 0b1.

Access to this field is **RO**.

SIZE, bits [13:8]

Size of counters, minus one. This field defines the size of the largest counter implemented by the Performance Monitors Unit.

From Armv8.0, the largest counter is 64-bits, so the value of this field is 0b111111.

This field is used by software to determine the spacing of the counters in the memory-map. From Armv8.0, the counters are a doubleword-aligned addresses.

Reads as 0b111111.

Access to this field is **RO**.

N, bits [7:0]

Number of counters accessible, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

N	Meaning
0x00	Only PMCCNTR_EL0 accessible.
0x01..0x20	Number of counters accessible, 2 to 33, minus one.

All other values are reserved.

If FEAT_PMUv3_ICNTR is implemented, then the value 0x00 is not permitted.

The count includes:

- The cycle counter, PMCCNTR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, the Instruction Counter, PMICNTR_EL0.

When FEAT_PMUv3_EXTPMN is implemented and the access to this register is not a Most secure access, the following apply:

- If FEAT_PMUv3_ICNTR is not implemented, this field reads as the Effective value of PMCCR.EPMN.
- If FEAT_PMUv3_ICNTR is implemented, this field reads as the Effective value of PMCCR.EPMN plus one.

Otherwise, the following apply:

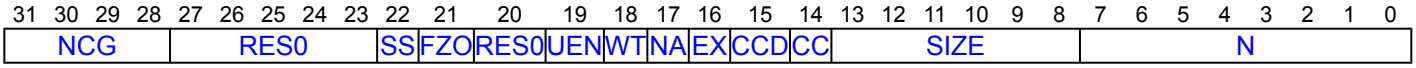
- If FEAT_PMUv3_ICNTR is not implemented, this field reads as the number of event counters implemented.
- If FEAT_PMUv3_ICNTR is implemented, this field reads as the number of event counters implemented plus one.

For example, if PMCFGR.N == 0x07 then:

- There are eight counters in total.
- If FEAT_PMUv3_ICNTR is not implemented, this comprises 7 event counters and the cycle counter.
- If FEAT_PMUv3_ICNTR is implemented, this comprises 6 event counters, the cycle counter, and the instruction counter.

Access to this field is **RO**.

Otherwise:



NCG, bits [31:28]

Defines the number of counter groups implemented, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCG	Meaning
0b0000	One counter group implemented.
0b0001	Two counter groups implemented.

All other values are reserved.

FEAT_PMUv3_ICNTR implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

Bits [27:23]

Reserved, RES0.

SS, bit [22]

Snapshot supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SS	Meaning
0b0	Snapshot mechanism not supported. The locations 0x600-0x7FC and 0xE30-0xE3C are IMPLEMENTATION DEFINED.
0b1	Snapshot mechanism supported. If FEAT_PMUv3_SS is implemented, then the following registers are implemented: • PMEVCNTSVR<n>_EL1. • PMCCNTSVR_EL1. • If FEAT_PMUv3_ICNTR is implemented, PMICNTSVR_EL1. • PMSSCR_EL1. Otherwise, locations 0x600-0x7FC and 0xE30-0xE3C contain IMPLEMENTATION DEFINED snapshot registers.

FEAT_PMUv3_SS implements the functionality identified by the value 1.

If FEAT_PMUv3_SS is not implemented, a PMU might include an IMPLEMENTATION DEFINED snapshot mechanism, including one using the IMPLEMENTATION DEFINED registers `0x600-0x7FC` and `0xE30-0xE3C`.

Access to this field is **RO**.

FZO, bit [21]

Freeze-on-overflow supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FZO	Meaning
`0b0`	Freeze-on-overflow mechanism is not supported. PMCR_EL0.FZO is RES0.
`0b1`	Freeze-on-overflow mechanism is supported. PMCR_EL0.FZO is RW.

FEAT_PMUv3p7 implements the functionality added by the value `0b1`.

From Armv8.7, if FEAT_PMUv3 is implemented, the only permitted value is `0b1`.

Access to this field is **RO**.

Bit [20]

Reserved, RES0.

UEN, bit [19]

User-mode Enable Register supported. PMUSERENR_EL0 is not visible in the external debug interface, so this bit is RAZ.

Reads as `0b0`.

Access to this field is **RO**.

WT, bit [18]

This feature is not supported, so this bit is RAZ.

Reads as `0b0`.

Access to this field is **RO**.

NA, bit [17]

This feature is not supported, so this bit is RAZ.

Reads as `0b0`.

Access to this field is **RO**.

EX, bit [16]

Export supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EX	Meaning
`0b0`	PMCR_EL0.X is RES0.
`0b1`	PMCR_EL0.X is read/write.

Access to this field is **RO**.

CCD, bit [15]

Cycle counter has prescale.

This is RES1 if AArch32 is supported, and RAZ otherwise.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCD	Meaning
0b0	PMCR_EL0.D is RES0.
0b1	PMCR_EL0.D is read/write.

Access to this field is **RO**.

CC, bit [14]

Dedicated cycle counter (counter 31) supported.

Reads as 0b1.

Access to this field is **RO**.

SIZE, bits [13:8]

Size of counters, minus one. This field defines the size of the largest counter implemented by the Performance Monitors Unit.

From Armv8.0, the largest counter is 64-bits, so the value of this field is 0b111111.

This field is used by software to determine the spacing of the counters in the memory-map. From Armv8.0, the counters are a doubleword-aligned addresses.

Reads as 0b111111.

Access to this field is **RO**.

N, bits [7:0]

Number of counters accessible, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

N	Meaning
0x00	Only PMCCNTR_EL0 accessible.
0x01..0x20	Number of counters accessible, 2 to 33, minus one.

All other values are reserved.

If FEAT_PMUv3_ICNTR is implemented, then the value 0x00 is not permitted.

The count includes:

- The cycle counter, PMCCNTR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, the Instruction Counter, PMICNTR_EL0.

When FEAT_PMUv3_EXTPMN is implemented and the access to this register is not a Most secure access, the following apply:

- If FEAT_PMUv3_ICNTR is not implemented, this field reads as the Effective value of PMCCR.EPMN.
- If FEAT_PMUv3_ICNTR is implemented, this field reads as the Effective value of PMCCR.EPMN plus one.

Otherwise, the following apply:

- If FEAT_PMUv3_ICNTR is not implemented, this field reads as the number of event counters implemented.
- If FEAT_PMUv3_ICNTR is implemented, this field reads as the number of event counters implemented plus one.

For example, if PMCFGR.N == 0x07 then:

- There are eight counters in total.

- If FEAT_PMUv3_ICNTR is not implemented, this comprises 7 event counters and the cycle counter.
- If FEAT_PMUv3_ICNTR is implemented, this comprises 6 event counters, the cycle counter, and the instruction counter.

Access to this field is **RO**.

Accessing PMCFGR

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented

[63:0] Accessible at offset `0xE00` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

When FEAT_PMUv3_EXT32 is implemented

[31:0] Accessible at offset `0xE00` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCGCR0, Counter Group Configuration Register 0

The PMCGCR0 characteristics are:

Purpose

Encodes the number of counters accessible.

Configuration

This register is present only when FEAT_PMUv3_ICNTR is implemented. Otherwise, direct accesses to PMCGCR0 are RES0.

PMCGCR0 is in the Core power domain.

Attributes

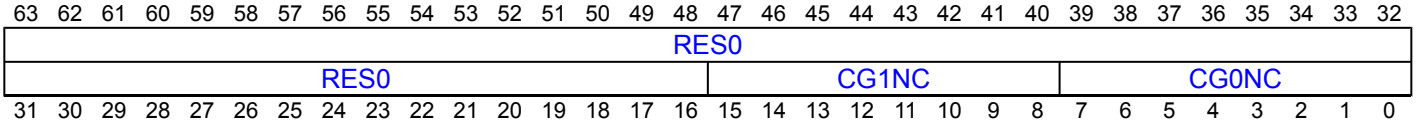
PMCGCR0 is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented:



Bits [63:16]

Reserved, RES0.

CG1NC, bits [15:8]

Number of counters in group 1, which comprises the instruction counter PMICNTR_EL0.

CG1NC	Meaning
0x01	PMICNTR_EL0 implemented.

Other values are reserved.

Access to this field is **RO**.

CG0NC, bits [7:0]

Number of counters in group 0, which comprises the event counters PMEVCNTR<n>_EL0 and the cycle counter PMCCNTR_EL0.

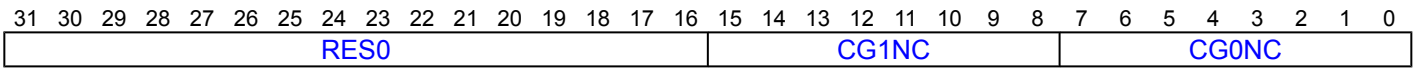
When FEAT_PMUv3_EXTPMN is implemented and the external access to this register is not a Most secure access, this field reads as the Effective value of PMCCR.EPMN plus one.

Otherwise, this field reads as the number of event counters implemented plus one.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:



Bits [31:16]

Reserved, RES0.

CG1NC, bits [15:8]

Number of counters in group 1, which comprises the instruction counter PMICNTR_EL0.

CG1NC	Meaning
0x01	PMICNTR_EL0 implemented.

Other values are reserved.

Access to this field is **RO**.

CG0NC, bits [7:0]

Number of counters in group 0, which comprises the event counters PMEVCNTR<n>_EL0 and the cycle counter PMCCNTR_EL0.

When FEAT_PMUv3_EXTPMN is implemented and the external access to this register is not a Most secure access, this field reads as the Effective value of PMCCR.EPMN plus one.

Otherwise, this field reads as the number of event counters implemented plus one.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMCGCR0

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT32 is implemented

[31:0] Accessible at offset 0xCE0 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

When FEAT_PMUv3_EXT64 is implemented

[63:0] Accessible at offset 0xCE0 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCID1SR, CONTEXTIDR_EL1 Sample Register

The PMCID1SR characteristics are:

Purpose

Contains the sampled value of CONTEXTIDR_EL1, captured on reading PMPCSR[31:0].

Configuration

This register is present only when FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented.

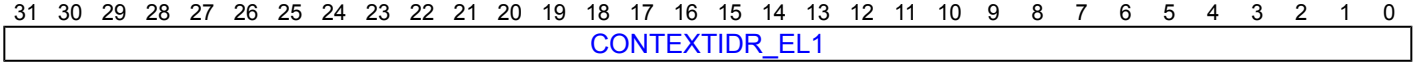
PMCID1SR is in the Core power domain.

Attributes

PMCID1SR is a 32-bit register.

This register is part of the PMU block.

Field descriptions



CONTEXTIDR_EL1, bits [31:0]

Context ID. The value of CONTEXTIDR that is associated with the most recent PMPCSR sample. When the most recent PMPCSR sample is generated:

- If EL1 is using AArch64, then the Context ID is sampled from CONTEXTIDR_EL1.
- If EL1 is using AArch32, then the Context ID is sampled from CONTEXTIDR.
- If EL3 is implemented and is using AArch32, then CONTEXTIDR is a banked register and this register samples the current banked copy of CONTEXTIDR for the Security state that is associated with the most recent PMPCSR sample.

Because the value written to this register is an indirect read of CONTEXTIDR, it is CONSTRAINED UNPREDICTABLE whether this register is set to the original or new value if PMPCSR samples:

- An instruction that writes to CONTEXTIDR.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCID1SR

If FEAT_PCSRv8p2 and FEAT_PMUv3_EXT64 are implemented, then the same content is present in the same locations, and can be accessed using PMCCIDSR[31:0] or PMCVIDSR[31:0].

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

Accesses to this register use the following encodings:

Accessible at offset `0x208` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.

- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Accessible at offset `0x228` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCID2SR, CONTEXTIDR_EL2 Sample Register

The PMCID2SR characteristics are:

Purpose

Contains the sampled value of CONTEXTIDR_EL2, captured on reading PMPCSR[31:0].

Configuration

This register is present only when FEAT_PMUv3_EXT32 is implemented and FEAT_PCSRv8p2 is implemented.

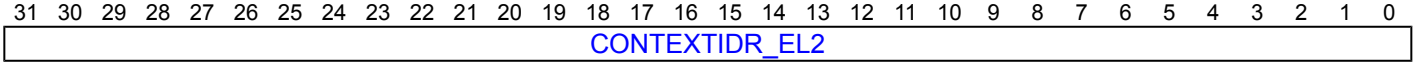
PMCIDR2SR is in the Core power domain.

Attributes

PMCID2SR is a 32-bit register.

This register is part of the PMU block.

Field descriptions



CONTEXTIDR_EL2, bits [31:0]

Context ID. The value of CONTEXTIDR_EL2 that is associated with the most recent PMPCSR sample. When the most recent PMPCSR sample is generated:

- If the PE is not executing at EL3, EL2 is using AArch64, and EL2 is enabled in the current Security state, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- Otherwise, this field is set to an UNKNOWN value.

Because the value written to this field is an indirect read of CONTEXTIDR_EL2, it is CONSTRAINED UNPREDICTABLE whether this field is set to the original or new value if PMPCSR samples:

- An instruction that writes to CONTEXTIDR_EL2.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCID2SR

If FEAT_PMUv3_EXT64 is implemented, then the same content is present in the same location, and can be accessed using PMCCIDSR[63:32].

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

Accesses to this register use the following encodings:

Accessible at offset `0x22C` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.

- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCIDR0, Performance Monitors Component Identification Register 0

The PMCIDR0 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Component Identification scheme'.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMCIDR0. Otherwise, direct accesses to PMCIDR0 are RES0.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

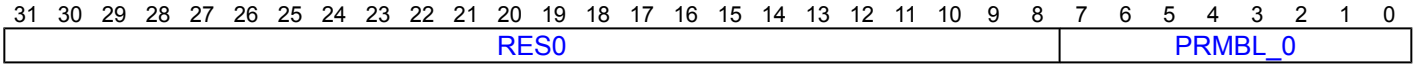
This register is required for CoreSight compliance.

Attributes

PMCIDR0 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Preamble.

Reads as `0x0D`.

Access to this field is **RO**.

Accessing PMCIDR0

Accesses to this register use the following encodings:

Accessible at offset `0xFF0` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCIDR1, Performance Monitors Component Identification Register 1

The PMCIDR1 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Component Identification scheme'.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMCIDR1. Otherwise, direct accesses to PMCIDR1 are RES0.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

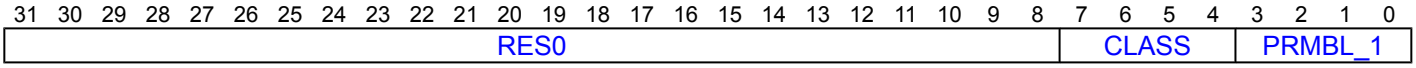
This register is required for CoreSight compliance.

Attributes

PMCIDR1 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

CLASS	Meaning
0b1001	CoreSight component.

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

Access to this field is **RO**.

PRMBL_1, bits [3:0]

Preamble.

Reads as 0b0000.

Access to this field is **RO**.

Accessing PMCIDR1

Accesses to this register use the following encodings:

Accessible at offset `0xFF4` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCIDR2, Performance Monitors Component Identification Register 2

The PMCIDR2 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Component Identification scheme'.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMCIDR2. Otherwise, direct accesses to PMCIDR2 are RES0.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

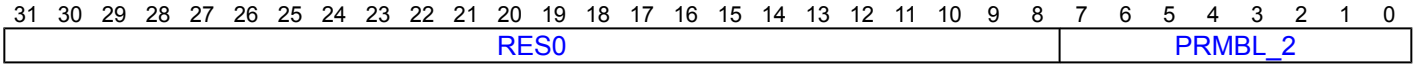
This register is required for CoreSight compliance.

Attributes

PMCIDR2 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Preamble.

Reads as `0x05`.

Access to this field is **RO**.

Accessing PMCIDR2

Accesses to this register use the following encodings:

Accessible at offset `0xFF8` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCIDR3, Performance Monitors Component Identification Register 3

The PMCIDR3 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Component Identification scheme'.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMCIDR3. Otherwise, direct accesses to PMCIDR3 are RES0.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

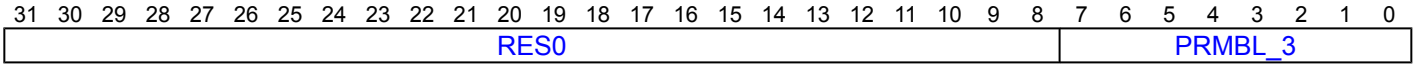
This register is required for CoreSight compliance.

Attributes

PMCIDR3 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Preamble.

Reads as `0xB1`.

Access to this field is **RO**.

Accessing PMCIDR3

Accesses to this register use the following encodings:

Accessible at offset `0xFFC` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTEN, Performance Monitors Count Enable register

The PMCNTEN characteristics are:

Purpose

Enables the Cycle Count Register, PMCCNTR_EL0, and any implemented event counters PMEVCNTR<n>.

Configuration

External register PMCNTEN bits [63:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[63:0].

External register PMCNTEN bits [63:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[63:0].

External register PMCNTEN bits [31:0] are architecturally mapped to AArch32 System register PMCNTENSET[31:0].

External register PMCNTEN bits [31:0] are architecturally mapped to AArch32 System register PMCNTENCLR[31:0].

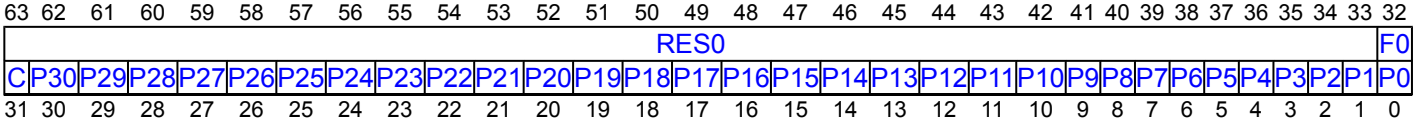
This register is present only when FEAT_PMUv3_EXT64 is implemented. Otherwise, direct accesses to PMCNTEN are RES0.

Attributes

PMCNTEN is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

PMICNTR_EL0 counter enable. Enables the instruction counter.

F0	Meaning
0b0	PMICNTR_EL0 disabled.
0b1	PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 enable. Enables the cycle counter register. Possible values are:

C	Meaning
0b0	PMCCNTR_EL0 is disabled.
0b1	PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

P<m>, bit [m], for m = 30 to 0

Event counter enable for PMEVCNTR<m>_EL0.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 is disabled.
0b1	PMEVCNTR<m>_EL0 is enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).
  - !IsMostSecureAccess(addrdesc).
- Otherwise, access to this field is **RW**.

Accessing PMCNTEN

Accesses to this register use the following encodings:

Accessible at offset 0xC10 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTENCLR_EL0, Performance Monitors Count Enable Clear Register

The PMCNTENCLR_EL0 characteristics are:

Purpose

Allows software to disable the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which counters are enabled.

Configuration

External register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3p9 is not implemented, and FEAT_PMUv3_ICNTR is not implemented.

External register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3p9 is not implemented, and FEAT_PMUv3_ICNTR is not implemented.

External register PMCNTENCLR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[63:0] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMCNTENCLR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[63:0] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENCLR[31:0].

External register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENSET[31:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMCNTENCLR_EL0 are RES0.

PMCNTENCLR_EL0 is in the Core power domain.

Attributes

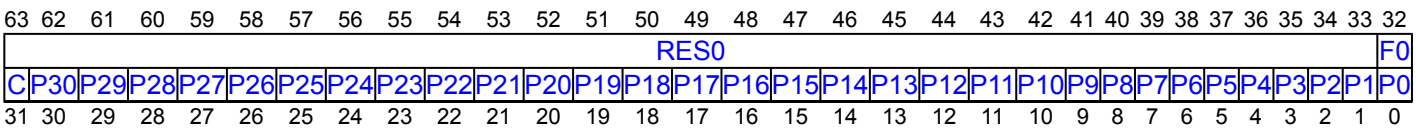
PMCNTENCLR_EL0 is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented:



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

PMICNTR_EL0 disable. On writes, allows software to disable PMICNTR_EL0. On reads, returns the PMICNTR_EL0 enable status.

F0	Meaning
0b0	PMICNTR_EL0 disabled.
0b1	PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 disable. On writes, allows software to disable PMCCNTR_EL0. On reads, returns the PMCCNTR_EL0 enable status.

C	Meaning
0b0	PMCCNTR_EL0 disabled.
0b1	PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

PMEVCNTR<m>_EL0 disable. On writes, allows software to disable PMEVCNTR<m>_EL0. On reads, returns the PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 disabled.
0b1	PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).

        - !IsMostSecureAccess(addrdesc).
    - When SoftwareLockStatus(), access to this field is **RO**.
    - Otherwise, access to this field is **W1C**.

Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

C, bit [31]

PMCCNTR_EL0 disable. On writes, allows software to disable PMCCNTR_EL0. On reads, returns the PMCCNTR_EL0 enable status.

C	Meaning
0b0	PMCCNTR_EL0 disabled.
0b1	PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

PMEVCNTR<m>_EL0 disable. On writes, allows software to disable PMEVCNTR<m>_EL0. On reads, returns the PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 disabled.
0b1	PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
    - FEAT_PMUv3_EXTPMN is implemented.
    - m >= UInt(EffectiveEPMN()).
    - !IsMostSecureAccess(addrdesc).
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

Accessing PMCNTENCLR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented

[63:0] Accessible at offset 0xC20 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXT32 is implemented and SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented

[31:0] Accessible at offset `0xC20` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTENSET_EL0, Performance Monitors Count Enable Set Register

The PMCNTENSET_EL0 characteristics are:

Purpose

Allows software to enable the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which counters are enabled.

Configuration

External register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3p9 is not implemented, and FEAT_PMUv3_ICNTR is not implemented.

External register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3p9 is not implemented, and FEAT_PMUv3_ICNTR is not implemented.

External register PMCNTENSET_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[63:0] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMCNTENSET_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[63:0] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENCLR[31:0].

External register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENSET[31:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMCNTENSET_EL0 are RES0.

PMCNTENSET_EL0 is in the Core power domain.

Attributes

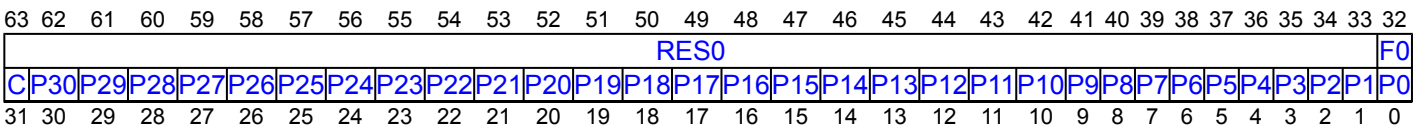
PMCNTENSET_EL0 is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented:



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

PMICNTR_EL0 enable. On writes, allows software to enable PMICNTR_EL0. On reads, returns the PMICNTR_EL0 enable status.

F0	Meaning
0b0	PMICNTR_EL0 disabled.
0b1	PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

Otherwise:

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 enable. On writes, allows software to enable PMCCNTR_EL0. On reads, returns the PMCCNTR_EL0 enable status.

C	Meaning
0b0	PMCCNTR_EL0 disabled.
0b1	PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

PMEVCNTR<m>_EL0 enable. On writes, allows software to enable PMEVCNTR<m>_EL0. On reads, returns the PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 disabled.
0b1	PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).

        - !IsMostSecureAccess(addrdesc).
    - When SoftwareLockStatus(), access to this field is **RO**.
    - Otherwise, access to this field is **W1S**.

Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

C, bit [31]

PMCCNTR_EL0 enable. On writes, allows software to enable PMCCNTR_EL0. On reads, returns the PMCCNTR_EL0 enable status.

C	Meaning
0b0	PMCCNTR_EL0 disabled.
0b1	PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

PMEVCNTR<m>_EL0 enable. On writes, allows software to enable PMEVCNTR<m>_EL0. On reads, returns the PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 disabled.
0b1	PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
    - FEAT_PMUv3_EXTPMN is implemented.
    - m >= UInt(EffectiveEPMN()).
    - !IsMostSecureAccess(addrdesc).
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

Accessing PMCNTENSET_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented

[63:0] Accessible at offset 0xC00 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXT32 is implemented and SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented

[31:0] Accessible at offset `0xC00` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCR_EL0, Performance Monitors Control Register

The PMCR_EL0 characteristics are:

Purpose

Configures and controls the Performance Monitors counters.

Configuration

External register PMCR_EL0 bits [63:32] are architecturally mapped to AArch64 System register PMCR_EL0[63:32] when FEAT_PMUv3_EXT64 is implemented.

External register PMCR_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMCR_EL0[31:0].

External register PMCR_EL0 bits [10:0] are architecturally mapped to AArch32 System register PMCR[10:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMCR_EL0 are RES0.

PMCR_EL0 is in the Core power domain.

This register is only partially mapped to the internal PMCR System register. An external agent must use other means to discover the information held in PMCR[31:11], such as accessing PMCFGR and the ID registers.

Attributes

PMCR_EL0 is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															FZS
RAZ/WI																					RES0	FZO	RES0	LP	LC	DP	X	D	C	P	E
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:33]

Reserved, RES0.

FZS, bit [32]

When FEAT_SPEv1p2 is implemented:

Freeze-on-SPE event. Stop counters when PMBLIMITR_EL1.{PMFZ,E} is {1,1} and profiling is stopped.

FZS	Meaning
0b0	Do not freeze on a Statistical Profiling Buffer Management event.
0b1	Affected counters do not count following a Statistical Profiling Buffer Management event.

The pseudocode function SPEProfilingStopped describes when profiling is stopped.

The counters affected by this field are:

- The event counters in the first range.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- If FEAT_SPE_DPFZS is implemented and PMCR_EL0.DP is 1, the cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

When FEAT_SPE_DPFZS is not implemented or PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA32 is implemented, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [31:11]

Reserved, RAZ/WI.

Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register.

Bit [10]

Reserved, RES0.

FZO, bit [9]

When FEAT_PMUv3p7 is implemented:

Freeze-on-overflow. Stop event counters on overflow.

FZO	Meaning
`0b0`	Do not freeze on overflow.
`0b1`	Affected counters do not count when any of the following applies: • For any event counter PMEVCNTR<m>_EL0 in the first range, PMOVSCLR_EL0[m] is 1, and either FEAT_SEBEP is not implemented or PMEVTYPER<m>_EL0.SYNC is 0. • FEAT_PMUv3_ICNTR is implemented, PMOVSCLR_EL0.F0 is 1, and either FEAT_SEBEP is not implemented or PMICFILTR_EL0.SYNC is 0.

The counters affected by this field are:

- The event counters in the first range.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- If PMCR_EL0.DP is 1, the cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

LP, bit [7]

When FEAT_PMUv3p5 is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by PMOVSCLR_EL0.P[n].

LP	Meaning
0b0	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[31:0].
0b1	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[63:0].

When FEAT_EBEP is implemented and the PMU Profiling exception is enabled, the Effective value of this field is 1.

The counters affected by this field are the event counters in the first range. For more information about event counter ranges, see MDCR_EL2.HPMN.

Other event counters and PMCCNTR_EL0 are not affected by this field.

When FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0 is not affected by this field.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LC, bit [6]

When FEAT_AA32 is implemented:

Long cycle counter enable. Determines when unsigned overflow is recorded by PMOVSCLR_EL0.C.

LC	Meaning
0b0	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[31:0].
0b1	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[63:0].

When FEAT_EBEP is implemented and the PMU Profiling exception is enabled, the Effective value of this field is 1.

Arm deprecates use of PMCR_EL0.LC = 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

DP, bit [5]

When EL3 is implemented or (FEAT_PMUv3p1 is implemented and EL2 is implemented):

Disable cycle counter when event counting is prohibited.

DP	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is disabled in prohibited regions and when event counting is frozen: • If FEAT_PMUv3p1 is implemented, EL2 is implemented, and MDCR_EL2.HPMD or HDCR.HPMD is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL2. • If FEAT_SPE_DPFZS is implemented and event counting is frozen by PMCR_EL0.FZS, then cycle counting by PMCCNTR_EL0 is disabled. • If FEAT_PMUv3p7 is implemented and event counting is frozen by PMCR_EL0.FZO, then cycle counting by PMCCNTR_EL0 is disabled. • If FEAT_PMUv3p7 is implemented, EL3 is implemented and using AArch64, and MDCR_EL3.MPMX is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL3. • If EL3 is implemented, MDCR_EL3.SPME or SDCR.SPME is 0, and one of FEAT_PMUv3p7 is not implemented, EL3 is using AArch32, or MDCR_EL3.MPMX is 0, then cycle counting by PMCCNTR_EL0 is disabled at EL3 and in Secure state.

The conditions when this field disables the cycle counter are the same as when event counting by an event counter in the first range is prohibited or frozen. For more information about event counter ranges, see MDCR_EL2.HPMN.

If FEAT_PMUv3p7 and FEAT_SPEv1p2 are implemented, meaning PMCR_EL0.FZS is implemented, and FEAT_SPE_DPFZS is not implemented, then cycle counting by PMCCNTR_EL0 is not affected by PMCR_EL0.FZS.

For more information, see 'Prohibiting event and cycle counting'.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

X, bit [4]

When the implementation includes a PMU event export bus:

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.

X	Meaning
0b0	Do not export events.
0b1	Export events where not prohibited.

This field enables the exporting of events over an IMPLEMENTATION DEFINED PMU event export bus to another device.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

If FEAT_ETE is implemented, this field does not affect the use of PMU events as an External Input by the trace unit.

If FEAT_ETMv4 is implemented, this field does affect the use of PMU events as an External Input by the trace unit.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

D, bit [3]

When FEAT_AA32 is implemented:

Clock divider.

D	Meaning
0b0	When enabled, PMCCNTR_EL0 counts every clock cycle.
0b1	When enabled, PMCCNTR_EL0 counts once every 64 clock cycles.

If the Effective value of PMCR_EL0.LC is 1, then this field is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR_EL0.D = 1.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

C, bit [2]

Cycle counter reset. The effects of writing to this field are:

C	Meaning
0b0	No action.
0b1	Reset PMCCNTR_EL0 to zero.

Note

Resetting PMCCNTR_EL0 does not change the cycle counter overflow field. The value of PMCR_EL0.LC is ignored, and bits [63:0] of the cycle counter are reset.

Access to this field is **WO/RAZ**.

P, bit [1]

Event counter reset.

P	Meaning
0b0	No action.
0b1	Reset all affected event counters PMEVCNTR<n>_EL0 to zero.

The event counters affected by this field are:

- If FEAT_PMUv3_EXTPMN is implemented and the access to this register is a Most secure access, all event counters.
- Otherwise, only event counters in the first and second ranges.

Writes to this field do not affect other event counters, the cycle counter PMCCNTR_EL0, or the instruction counter PMICNTR_EL0.

For more information about event counter ranges, see MDCR_EL2.HPMN.

Note

Resetting the event counters does not change the event counter overflow fields. If FEAT_PMUv3p5 is implemented, the values of MDCR_EL2.HLP or HDCR.HLP and PMCR_EL0.LP are ignored, and bits [63:0] of all affected event counters are reset.

Access to this field is **WO/RAZ**.

E, bit [0]

Enable.

E	Meaning
0b0	Affected counters are disabled and do not count.
0b1	Affected counters are enabled by PMCNTENSET_EL0.

The counters affected by this field are:

- The event counters in the first range. For more information about event counter ranges, see MDCR_EL2.HPMN.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- The cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RAZ/WI																					RES0	FZO	RES0	LP	LC	DP	X	D	C	P	E

Bits [31:11]

Reserved, RAZ/WI.

Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register.

Bit [10]

Reserved, RES0.

FZO, bit [9]

When FEAT_PMUv3p7 is implemented:

Freeze-on-overflow. Stop event counters on overflow.

FZO	Meaning
0b0	Do not freeze on overflow.
0b1	Affected counters do not count when any of the following applies: • For any event counter PMEVCNTR<m>_EL0 in the first range, PMOVSCLR_EL0[m] is 1, and either FEAT_SEBEP is not implemented or PMEVTYPER<m>_EL0.SYNC is 0. • FEAT_PMUv3_ICNTR is implemented, PMOVSCLR_EL0.F0 is 1, and either FEAT_SEBEP is not implemented or PMICFILTR_EL0.SYNC is 0.

The counters affected by this field are:

- The event counters in the first range.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

- If PMCR_EL0.DP is 1, the cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

LP, bit [7]

When FEAT_PMUv3p5 is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by PMOVSCLR_EL0.P[n].

LP	Meaning
0b0	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[31:0].
0b1	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[63:0].

When FEAT_EBEP is implemented and the PMU Profiling exception is enabled, the Effective value of this field is 1.

The counters affected by this field are the event counters in the first range. For more information about event counter ranges, see MDCR_EL2.HPMN.

Other event counters and PMCCNTR_EL0 are not affected by this field.

When FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0 is not affected by this field.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LC, bit [6]

When FEAT_AA32 is implemented:

Long cycle counter enable. Determines when unsigned overflow is recorded by PMOVSCLR_EL0.C.

LC	Meaning
0b0	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[31:0].
0b1	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[63:0].

When FEAT_EBEP is implemented and the PMU Profiling exception is enabled, the Effective value of this field is 1.

Arm deprecates use of PMCR_EL0.LC = 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

DP, bit [5]

When EL3 is implemented or (FEAT_PMUv3p1 is implemented and EL2 is implemented):

Disable cycle counter when event counting is prohibited.

DP	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is disabled in prohibited regions and when event counting is frozen: • If FEAT_PMUv3p1 is implemented, EL2 is implemented, and MDCR_EL2.HPMD or HDCR.HPMD is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL2. • If FEAT_SPE_DPFZS is implemented and event counting is frozen by PMCR_EL0.FZS, then cycle counting by PMCCNTR_EL0 is disabled. • If FEAT_PMUv3p7 is implemented and event counting is frozen by PMCR_EL0.FZO, then cycle counting by PMCCNTR_EL0 is disabled. • If FEAT_PMUv3p7 is implemented, EL3 is implemented and using AArch64, and MDCR_EL3.MPMX is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL3. • If EL3 is implemented, MDCR_EL3.SPME or SDCR.SPME is 0, and one of FEAT_PMUv3p7 is not implemented, EL3 is using AArch32, or MDCR_EL3.MPMX is 0, then cycle counting by PMCCNTR_EL0 is disabled at EL3 and in Secure state.

The conditions when this field disables the cycle counter are the same as when event counting by an event counter in the first range is prohibited or frozen. For more information about event counter ranges, see MDCR_EL2.HPMN.

If FEAT_PMUv3p7 and FEAT_SPEv1p2 are implemented, meaning PMCR_EL0.FZS is implemented, and FEAT_SPE_DPFZS is not implemented, then cycle counting by PMCCNTR_EL0 is not affected by PMCR_EL0.FZS.

For more information, see 'Prohibiting event and cycle counting'.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

X, bit [4]

When the implementation includes a PMU event export bus:

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.

X	Meaning
0b0	Do not export events.
0b1	Export events where not prohibited.

This field enables the exporting of events over an IMPLEMENTATION DEFINED PMU event export bus to another device.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

If FEAT_ETE is implemented, this field does not affect the use of PMU events as an External Input by the trace unit.

If FEAT_ETMv4 is implemented, this field does affect the use of PMU events as an External Input by the trace unit.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

D, bit [3]

When FEAT_AA32 is implemented:

Clock divider.

D	Meaning
`0b0`	When enabled, PMCCNTR_EL0 counts every clock cycle.
`0b1`	When enabled, PMCCNTR_EL0 counts once every 64 clock cycles.

If the Effective value of PMCR_EL0.LC is 1, then this field is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR_EL0.D = 1.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA64 is not implemented, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

C, bit [2]

Cycle counter reset. The effects of writing to this field are:

C	Meaning
`0b0`	No action.
`0b1`	Reset PMCCNTR_EL0 to zero.

Note

Resetting PMCCNTR_EL0 does not change the cycle counter overflow field. The value of PMCR_EL0.LC is ignored, and bits [63:0] of the cycle counter are reset.

Access to this field is **WO/RAZ**.

P, bit [1]

Event counter reset.

P	Meaning
0b0	No action.
0b1	Reset all affected event counters PMEVCNTR<n>_EL0 to zero.

The event counters affected by this field are:

- If FEAT_PMUv3_EXTPMN is implemented and the access to this register is a Most secure access, all event counters.
- Otherwise, only event counters in the first and second ranges.

Writes to this field do not affect other event counters, the cycle counter PMCCNTR_EL0, or the instruction counter PMICNTR_EL0.

For more information about event counter ranges, see MDCR_EL2.HPMN.

Note

Resetting the event counters does not change the event counter overflow fields. If FEAT_PMUv3p5 is implemented, the values of MDCR_EL2.HLP or HDCR.HLP and PMCR_EL0.LP are ignored, and bits [63:0] of all affected event counters are reset.

Access to this field is **WO/RAZ**.

E, bit [0]

Enable.

E	Meaning
0b0	Affected counters are disabled and do not count.
0b1	Affected counters are enabled by PMCNTENSET_EL0.

The counters affected by this field are:

- The event counters in the first range. For more information about event counter ranges, see MDCR_EL2.HPMN.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- The cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing PMCR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT32 is implemented

Accessible at offset 0xE04 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT64 is implemented

Accessible at offset `0xE10` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMDEVAFF, Performance Monitors Device Affinity register

The PMDEVAFF characteristics are:

Purpose

Copy of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the Performance Monitor component relates to.

Configuration

This register is present only when FEAT_PMUv3_EXT64 is implemented.

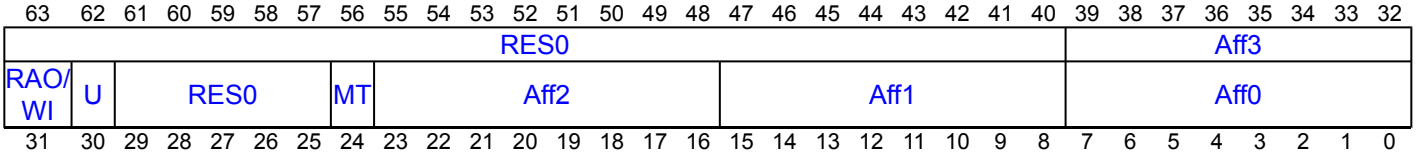
If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

PMDEVAFF is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [31]

Reserved, RAO/WI.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Access to this field is **RO**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is **RO**.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMDEVAFF

If FEAT_PMUv3_EXT32 is implemented, then the same content is present in the same locations, and can be accessed using PMDEVAFF0[31:0] and PMDEVAFF1[31:0].

Accesses to this register use the following encodings:

Accessible at offset 0xFA8 from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMDEVAFF0, Performance Monitors Device Affinity register 0

The PMDEVAFF0 characteristics are:

Purpose

Copy of the low half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the Performance Monitor component relates to.

Configuration

This register is present only when FEAT_PMUv3_EXT32 is implemented.

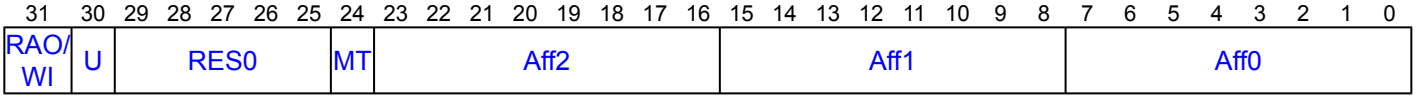
If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

PMDEVAFF0 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bit [31]

Reserved, RAO/WI.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Access to this field is **RO**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is **RO**.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMDEVAFF0

If FEAT_PMUv3_EXT64 is implemented, then the same content is present in the same location, and can be accessed using PMDEVAFF[31:0].

Accesses to this register use the following encodings:

Accessible at offset 0xFA8 from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMDEVAFF1, Performance Monitors Device Affinity register 1

The PMDEVAFF1 characteristics are:

Purpose

Copy of the high half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the Performance Monitor component relates to.

Configuration

This register is present only when FEAT_PMUv3_EXT32 is implemented.

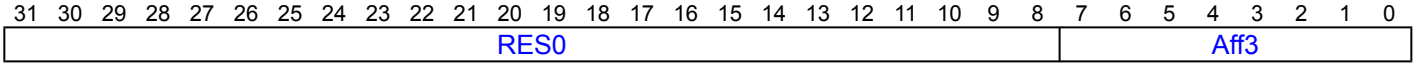
If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

PMDEVAFF1 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

Aff3, bits [7:0]

Affinity level 3. See the description of PMDEVAFF0.Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMDEVAFF1

If FEAT_PMUv3_EXT64 is implemented, then the same content is present in the same location, and can be accessed using PMDEVAFF[63:32].

Accesses to this register use the following encodings:

Accessible at offset `0xFAC` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMDEVARCH, Performance Monitors Device Architecture register

The PMDEVARCH characteristics are:

Purpose

Identifies the programmers' model architecture of the Performance Monitor component.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMDEVARCH are RES0.

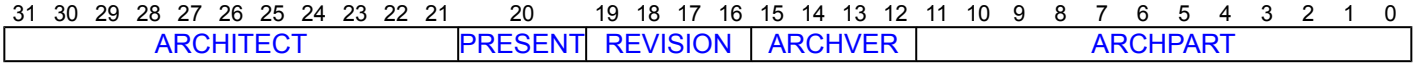
If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

PMDEVARCH is a 32-bit register.

This register is part of the PMU block.

Field descriptions



ARCHITECT, bits [31:21]

Defines the architect of the component. For Performance Monitors, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, `0b0100`.

Bits [27:21] are the JEP106 identification code, `0b0111011`.

Reads as `0b01000111011`.

Access to this field is **RO**.

PRESENT, bit [20]

DEVARCH present. Indicates that the PMDEVARCH register is present.

Reads as `0b1`.

Access to this field is **RO**.

REVISION, bits [19:16]

Defines the architecture revision. For architectures defined by Arm this is the minor revision.

For Performance Monitors, the revision defined by Armv8 is `0x0`.

All other values are reserved.

Reads as `0b0000`.

Access to this field is **RO**.

ARCHVER, bits [15:12]

When UInt(PMU.PMDEVARCH.ARCHPART) == 0xA16 or UInt(PMU.PMDEVARCH.ARCHPART) == 0xA26:

Architecture Version. Defines the architecture version of the component.

ARCHVER	Meaning
0b0010	Performance Monitors Extension version 3, PMUv3.

All other values are reserved.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHVER is PMDEVARCH.ARCHID[15:12].

Access to this field is **RO**.

Otherwise:

Architecture Version. Defines the architecture version of the component.

ARCHVER	Meaning
0b0000	PC Sample-based Profiling version 2, FEAT_PCSRv8p2.

All other values are reserved.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHVER is PMDEVARCH.ARCHID[15:12].

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ARCHPART	Meaning
0xA10	PC Sample-based Profiling, including the 32-bit programmers' model extension.
0xA16	Armv8-A PE performance monitors, including the 32-bit programmers' model extension.
0xA20	PC Sample-based Profiling, including the 64-bit programmers' model extension.
0xA26	Armv8-A PE performance monitors, including the 64-bit programmers' model extension.

FEAT_PMUv3_EXT32 implements the functionality described by the values 0xA10 and 0xA16.

FEAT_PMUv3_EXT64 implements the functionality described by the values 0xA20 and 0xA26.

The values 0xA10 and 0xA20 indicate that FEAT_PCSRv8p2 is implemented but the Performance Monitors Extension is not implemented.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHPART is PMDEVARCH.ARCHID[11:0].

Access to this field is **RO**.

Accessing PMDEVARCH

Accesses to this register use the following encodings:

Accessible at offset 0xFBC from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMDEVID, Performance Monitors Device ID register

The PMDEVID characteristics are:

Purpose

Provides information about features of the Performance Monitors implementation.

Configuration

This register is present only when (v8Ap2 or FEAT_PCSRv8p2 is implemented) and FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMDEVID are RES0.

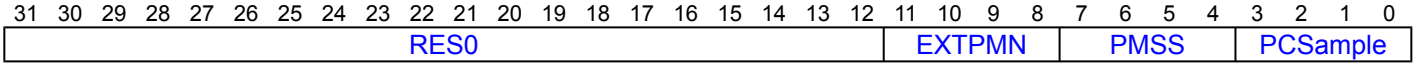
If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

PMDEVID is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:12]

Reserved, RES0.

EXTPMN, bits [11:8]

Reserving PMU event counters for external agents.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXTPMN	Meaning
0b0000	PMUv3 Extension for reserving PMU event counters for external agents not implemented.
0b0001	PMUv3 Extension for reserving PMU event counters for external agents implemented.

All other values are reserved.

FEAT_PMUv3_EXTPMN implements the functionality identified by the value 0b0001.

Access to this field is **RO**.

PMSS, bits [7:4]

PMU Snapshot extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMSS	Meaning
0b0000	PMU snapshot extension not implemented.
0b0001	PMU snapshot extension implemented.

All other values are reserved.

FEAT_PMUv3_SS implements the functionality identified by the value `0b0001`.

Access to this field is **RO**.

PCSample, bits [3:0]

Indicates the level of PC Sample-based Profiling support using Performance Monitors registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PCSample	Meaning
`0b0000`	PC Sample-based Profiling Extension is not implemented in the Performance Monitors register space.
`0b0001`	PC Sample-based Profiling Extension is implemented in the Performance Monitors register space.
`0b0010`	As `0b0001`, and adds support for PMPCSCTL.

All other values are reserved.

FEAT_PCSRv8p2 implements the functionality identified by the value `0b0001`.

FEAT_PCSRv8p9 implements the functionality identified by the value `0b0010`.

If FEAT_PCSRv8p2 is not implemented, then the only permitted value is `0b0000`.

From Armv8.2, when FEAT_PCSRv8p2 is implemented, the value `0b0000` is not permitted.

From Armv8.9, when FEAT_PCSRv8p9 is implemented, the value `0b0001` is not permitted.

Access to this field is **RO**.

Accessing PMDEVID

Accesses to this register use the following encodings:

Accessible at offset `0xFC8` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMDEVTYPE, Performance Monitors Device Type register

The PMDEVTYPE characteristics are:

Purpose

Indicates to a debugger that this component is part of a PE's performance monitor interface.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMDEVTYPE. Otherwise, direct accesses to PMDEVTYPE are RES0.

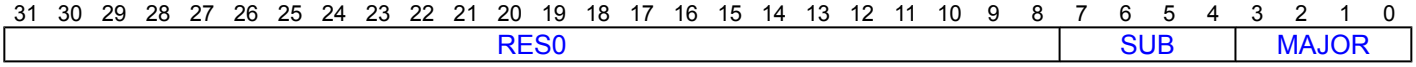
If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

PMDEVTYPE is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Subtype. Indicates this is a component within a PE.

Reads as `0b0001`.

Access to this field is **RO**.

MAJOR, bits [3:0]

Major type.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MAJOR	Meaning
`0b0000`	Unspecified.
`0b0110`	Performance monitor component.

FEAT_PMUv3 implements the functionality described by the value `0b0110`.

Access to this field is **RO**.

Accessing PMDEVTYPE

Accesses to this register use the following encodings:

Accessible at offset `0xFCC` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVCNTR<n>_EL0, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n>_EL0 characteristics are:

Purpose

Holds event counter <n>, which counts events, where <n> is 0 to 30.

Configuration

External register PMEVCNTR<n>_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMEVCNTR<n>_EL0[63:0] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p5 is implemented.

External register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMEVCNTR<n>_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p5 is not implemented.

External register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVCNTR<n>[31:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMEVCNTR<n>_EL0 are RES0.

PMEVCNTR<n>_EL0 is in the Core power domain.

Attributes

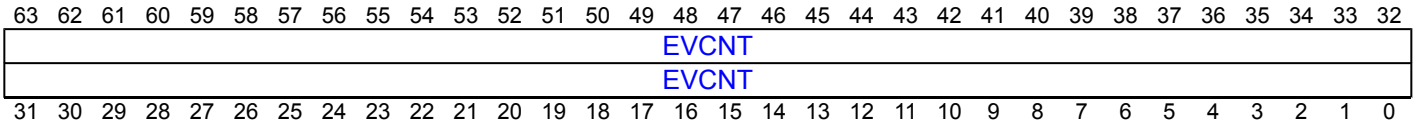
PMEVCNTR<n>_EL0 is a:

- 64-bit register when FEAT_PMUv3p5 is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3p5 is implemented:



EVCNT, bits [63:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

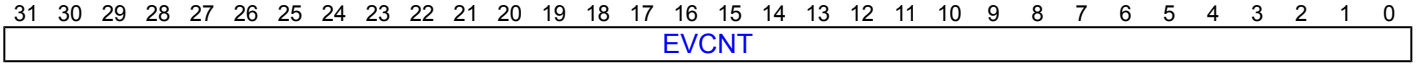
If the highest implemented Exception level is using AArch32, the optional external interface to the performance monitors is implemented, and the PMCR.LP and HDCR.HLP bits are RAZ/WI, then locations in the external interface to the performance monitors that map to PMEVCNTR<n>_EL0[63:32] return UNKNOWN values on reads.

If the implementation does not support AArch64, bits [63:32] of the event counters are not required to be implemented.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:



EVCNT, bits [31:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMEVCNTR<n>_EL0

If FEAT_PMUv3p5 is not implemented, when IsCorePowered(), DoubleLockStatus(), OSLockStatus() or !AllowExternalPMUAccess(), 32-bit accesses to `0x004+8×n` have a CONSTRAINED UNPREDICTABLE behavior.

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented

[63:0] Accessible at offset `0x000 + (8 * n)` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXTPMN is implemented, IsRange3Counter(n), and !IsMostSecureAccess(addrdesc), accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p5 is implemented

[63:0] Accessible at offset `0x000 + (8 * n)` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXTPMN is implemented, IsRange3Counter(n), and !IsMostSecureAccess(addrdesc), accesses to this register are **RAZ/WI**.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p5 is not implemented

[31:0] Accessible at offset `0x000 + (8 * n)` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVCNTSVR<n>_EL1, Performance Monitors Event Count Saved Value Registers, n = 0 - 30

The PMEVCNTSVR<n>_EL1 characteristics are:

Purpose

Captures the PMU Event counter <n>, PMEVCNTR<n>_EL0.

Configuration

External register PMEVCNTSVR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMEVCNTSVR<n>_EL1[63:0].

This register is present only when FEAT_PMUv3_SS is implemented. Otherwise, direct accesses to PMEVCNTSVR<n>_EL1 are RES0.

PMEVCNTSVR<n>_EL1 is in the Core power domain.

If event counter n is not implemented:

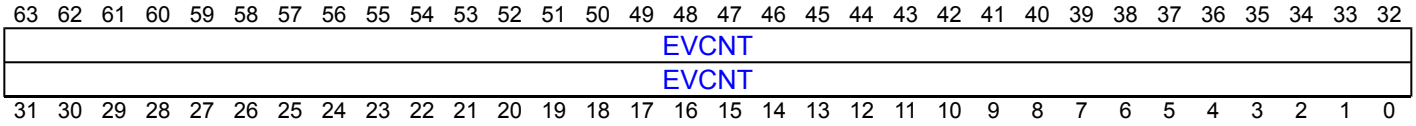
- When IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalPMUAccess(addrdesc), accesses are RES0.
- Otherwise, it is CONSTRAINED UNPREDICTABLE whether accesses to this register are RES0 or generate an error response.

Attributes

PMEVCNTSVR<n>_EL1 is a 64-bit register.

This register is part of the PMU block.

Field descriptions



EVCNT, bits [63:0]

Sampled Event Count. The value of PMEVCNTR<n>_EL0 at the last successful Capture event.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMEVCNTSVR<n>_EL1

Accesses to this register use the following encodings:

Accessible at offset `0x600 + (8 * n)` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMSSAccess(addrdesc), accesses to this register generate an error response.
- When FEAT_PMUv3_EXTPMN is implemented, IsRange3Counter(n), and !IsMostSecureAccess(addrdesc), accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVFILT2R<n>, Performance Monitors Event Filter Registers, n = 0 - 63

The PMEVFILT2R<n> characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED configuration controls for PMU counters.

Each PMEVFILT2R<n> register can provide additional configuration controls for a PMU counter, where:

- For values of n less than 31, if event counter n is implemented, then the controls are for PMU event counter <n>.
- For n equal to 31, the controls are for the cycle counter, PMCCNTR_EL0;
- For n equal to 32, if FEAT_PMUv3_ICNTR is implemented, the controls are for the instruction counter, PMICNTR_EL0;
- For all other values of n, PMEVFILT2R<n> is not implemented.

Although this mapping is recommended, it is not required and the function of each register is IMPLEMENTATION DEFINED.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMEVFILT2R<n>. Otherwise, direct accesses to PMEVFILT2R<n> are RES0.

PMEVFILT2R<n> is in the Core power domain.

If PMEVFILT2R<n> is not implemented:

- When IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalPMUAccess(), accesses are RES0.
- Otherwise, it is CONSTRAINED UNPREDICTABLE whether accesses to this register are RES0 or generate an error response.

Attributes

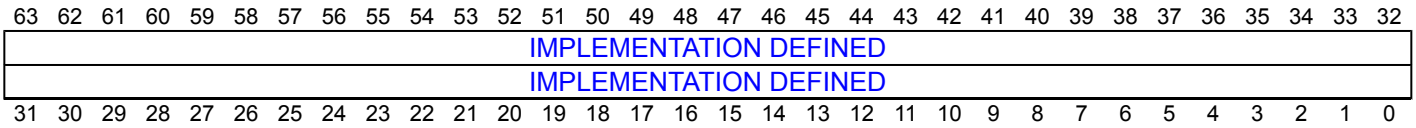
PMEVFILT2R<n> is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

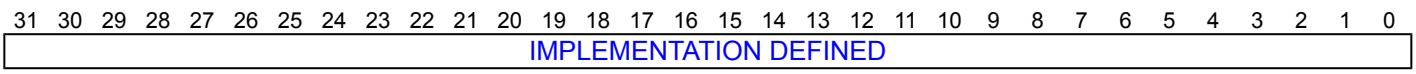
When FEAT_PMUv3_EXT64 is implemented:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Otherwise:



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing PMEVFILT2R<n>

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT32 is implemented

[31:0] Accessible at offset `0x800 + (4 * n)` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXTPMN is implemented, IsRange3Counter(n), and !IsMostSecureAccess(addrdesc), accesses to this register are **RAZ/WI**.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT64 is implemented

[63:0] Accessible at offset `0x800 + (8 * n)` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXTPMN is implemented, IsRange3Counter(n), and !IsMostSecureAccess(addrdesc), accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30

The PMEVTYPER<n>_EL0 characteristics are:

Purpose

Configures event counter n, where n is 0 to 30.

Configuration

External register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMEVTYPER<n>_EL0[31:0].

External register PMEVTYPER<n>_EL0 bits [63:32] are architecturally mapped to AArch64 System register PMEVTYPER<n>_EL0[63:32] when FEAT_PMUv3_TH is implemented, or FEAT_PMUv3p8 is implemented, or FEAT_PMUv3_EXT64 is implemented.

External register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVTYPER<n>[31:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMEVTYPER<n>_EL0 are RES0.

PMEVTYPER<n>_EL0 is in the Core power domain.

If event counter n is not implemented:

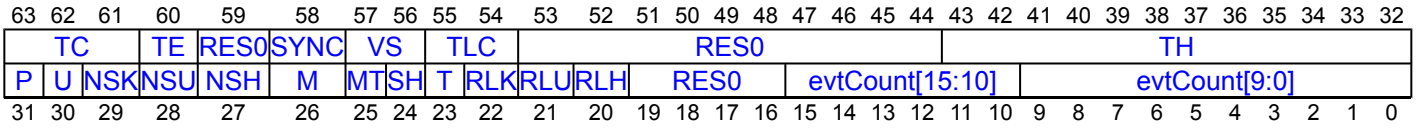
- When IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalPMUAccess(), accesses are RES0.
- Otherwise, it is CONSTRAINED UNPREDICTABLE whether accesses to this register are RES0 or generate an error response.

Attributes

PMEVTYPER<n>_EL0 is a 64-bit register.

This register is part of the PMU block.

Field descriptions



TC, bits [63:61]

When FEAT_PMUv3_TH is implemented, (FEAT_PMUv3_EDGE is not implemented or PMU.PMEVTYPER<n>_EL0.TE == 0), and (FEAT_PMUv3_TH2 is not implemented, or n is even, or PMU.PMEVTYPER<n>_EL0.TLC IN {0b0x}):

Threshold Control. Defines the threshold function. In the description of this field:

- V_B[n] is the value the event specified by PMEVTYPER<n>_EL0 would increment event counter n by on a processor cycle if the threshold function is disabled.
- For odd values of n, V[n-1] is the value that event counter n-1 increments by on the same processor cycle. V[n-1] is the result of applying the threshold and edge functions on event counter n-1. If event counter n-1 is disabled then V[n-1] is zero. V[n-1] is not defined for even values of n.
- TH[n] is the value of PMEVTYPER<n>_EL0.TH.

TC	Meaning
0b000	Not-equal. The counter increments by V_B[n] on each processor cycle when V_B[n] is not equal to TH[n].
0b001	Not-equal, count. The counter increments by 1 on each processor cycle when V_B[n] is not equal to TH[n].
0b010	Equals. The counter increments by V_B[n] on each processor cycle when V_B[n] is equal to TH[n].
0b011	Equals, count. The counter increments by 1 on each processor cycle when V_B[n] is equal to TH[n].
0b100	Greater-than-or-equal. The counter increments by V_B[n] on each processor cycle when V_B[n] is greater than or equal to TH[n].
0b101	Greater-than-or-equal, count. The counter increments by 1 on each processor cycle when V_B[n] is greater than or equal to TH[n].
0b110	Less-than. The counter increments by V_B[n] on each processor cycle when V_B[n] is less than TH[n].
0b111	Less-than, count. The counter increments by 1 on each processor cycle when V_B[n] is less than TH[n].

Comparisons treat V_B[n] and TH[n] as unsigned integer values.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC[2:1] is true:

- If PMEVTYPER<n>_EL0.TC[0] is 0, then the counter increments by V_B[n].
- If PMEVTYPER<n>_EL0.TC[0] is 1, then the counter increments by 1.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC[2:1] is false:

- If FEAT_PMUv3_TH2 is implemented, n is odd, and PMEVTYPER<n>_EL0.TLC is 0b01, then the counter increments by V[n-1].
- Otherwise, the counter does not increment.

If PMEVTYPER<n>_EL0.{TC, TLC, TH} are zero then the threshold function is disabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to '000'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When FEAT_PMUv3_TH2 is implemented, PMU.PMEVTYPER<n>_EL0.TE == 0, n is odd, and PMU.PMEVTYPER<n>_EL0.TLC == 0b10:

Threshold Control. Defines the threshold function. In the description of this field:

- V_B[n] is the value the event specified by PMEVTYPER<n>_EL0 would increment event counter n by on a processor cycle if the threshold function is disabled.
- V[n-1] is the value that event counter n-1 increments by on the same processor cycle. V[n-1] is the result of applying the threshold and edge functions on event counter n-1. If event counter n-1 is disabled then V[n-1] is zero.
- TH[n] is the value of PMEVTYPER<n>_EL0.TH.

TC	Meaning
0b000	Not-equal. The counter increments by V[n-1] on each processor cycle when V_B[n] is not equal to TH[n].
0b010	Equals. The counter increments by V[n-1] on each processor cycle when V_B[n] is equal to TH[n].
0b100	Greater-than-or-equal. The counter increments by V[n-1] on each processor cycle when V_B[n] is greater than or equal to TH[n].
0b110	Less-than. The counter increments by V[n-1] on each processor cycle when V_B[n] is less than TH[n].

All other values are reserved.

Comparisons treat V_B[n] and TH[n] as unsigned integer values.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is true, the counter increments by V[n-1].

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is false, the counter does not increment.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to `'000'`.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When FEAT_PMUv3_EDGE is implemented and PMU.PMEVTYPER<n>_EL0.TE == 1:

Threshold Control. Defines the threshold function. In the description of this field:

- V_B[n] is the value the event specified by PMEVTYPER<n>_EL0 would increment event counter n by on a processor cycle if the threshold function is disabled.
- For odd values of n, V[n-1] is the value that event counter n-1 increments by on the same processor cycle. V[n-1] is the result of applying the threshold and edge functions on event counter n-1. If event counter n-1 is disabled then V[n-1] is zero. V[n-1] is not defined for even values of n.
- TH[n] is the value of PMEVTYPER<n>_EL0.TH.

TC	Meaning
`0b001`	Equal to not-equal. The counter increments on each processor cycle when V_B[n] is not equal to TH[n] and V_B[n] was equal to TH[n] on the previous processor cycle.
`0b010`	Equal to/from not-equal. The counter increments on each processor cycle when either: • V_B[n] is not equal to TH[n] and V_B[n] was equal to TH[n] on the previous processor cycle. • V_B[n] is equal to TH[n] and V_B[n] was not equal to TH[n] on the previous processor cycle.
`0b011`	Not-equal to equal. The counter increments on each processor cycle when V_B[n] is equal to TH[n] and V_B[n] was not equal to TH[n] on the previous processor cycle.
`0b101`	Less-than to greater-than-or-equal. The counter increments on each processor cycle when V_B[n] is greater than or equal to TH[n] and V_B[n] was less than TH[n] on the previous processor cycle.
`0b110`	Less-than to/from greater-than-or-equal. The counter increments on each processor cycle when either: • V_B[n] is greater than or equal to TH[n] and V_B[n] was less than TH[n] on the previous processor cycle. • V_B[n] is less than TH[n] and V_B[n] was greater than or equal to TH[n] on the previous processor cycle.
`0b111`	Greater-than-or-equal to less-than. The counter increments on each processor cycle when V_B[n] is less than TH[n] and V_B[n] was greater than or equal to TH[n] on the previous processor cycle.

All other values are reserved.

Comparisons treat V_B[n] and TH[n] as unsigned integer values.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is true:

- If FEAT_PMUv3_TH2 is implemented, n is odd, and PMEVTYPER<n>_EL0.TLC is `0b10`, then the counter increments by V[n-1].
- Otherwise, the counter increments by 1.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is false, the counter does not increment.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to `'000'`.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TE, bit [60]

When FEAT_PMUv3_EDGE is implemented:

Threshold Edge. Enables the edge condition. When PMEVTYPER<n>_EL0.TE is 1, the event counter increments on cycles when the result of the threshold condition changes. See PMEVTYPER<n>_EL0.TC for more information.

TE	Meaning
0b0	Threshold edge condition disabled.
0b1	Threshold edge condition enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to '0'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [59]

Reserved, RES0.

SYNC, bit [58]

When FEAT_SEBEP is implemented:

Synchronous mode. Controls whether a PMU Profiling exception generated by the counter is synchronous or asynchronous.

SYNC	Meaning
0b0	Asynchronous PMU Profiling exception is enabled.
0b1	Synchronous PMU Profiling exception is enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VS, bits [57:56]

When FEAT_PMUv3_SME is implemented:

SVE mode filtering. Controls counting events in Streaming and Non-streaming SVE modes.

VS	Meaning
0b00	This mechanism has no effect on the filtering of events.
0b01	The PE does not count events in Streaming SVE mode.
0b10	The PE does not count events in Non-streaming SVE mode.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLC, bits [55:54]

When FEAT_PMUv3_TH2 is implemented and n is odd:

Threshold Linking Control. Extends PMEVTYPER<n>_EL0.TC with additional controls for event linking. See PMEVTYPER<n>_EL0.TC.

TLC	Meaning
0b00	Threshold linking disabled.
0b01	Threshold linking enabled. If the threshold condition described by PMEVTYPER<n>_EL0.TC is false, the counter increments by V[n-1]. Otherwise, the counter increments as described by PMEVTYPER<n>_EL0.TC.
0b10	Threshold linking enabled. If the threshold condition described by PMEVTYPER<n>_EL0.TC is true, the counter increments by V[n-1]. Otherwise, the counter does not increment.

All other values are reserved.

See PMEVTYPER<n>_EL0.TC for more information

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [53:44]

Reserved, RES0.

TH, bits [43:32]

When FEAT_PMUv3_TH is implemented:

Threshold value. Provides the unsigned value for the threshold function defined by PMEVTYPER<n>_EL0.TC.

If PMEVTYPER<n>_EL0.{TC, TH} are both zero and either FEAT_PMUv3_TH2 is not implemented or PMEVTYPER<n>_EL0.TLC is also zero, then the threshold function is disabled.

If PMU.PMMIR_EL1.THWIDTH is less than 12, then bits PMEVTYPER<n>_EL0.TH[11:UInt(PMU.PMMIR_EL1.THWIDTH)] are RES0. This accounts for the behavior when writing a value greater-than-or-equal-to $2^{UInt(PMU.PMMIR\_EL1.THWIDTH)}$.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to '000000000000'.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

P, bit [31]

EL1 filtering. Controls counting events in EL1.

P	Meaning
0b0	This mechanism has no effect on filtering of events.
0b1	The PE does not count events in EL1.

If Secure and Non-secure states are implemented, then counting events in Non-secure EL1 is further controlled by PMEVTYPER<n>_EL0.NSK.

If FEAT_RME is implemented, then counting events in Realm EL1 is further controlled by PMEVTYPER<n>_EL0.RLK.

If EL3 is implemented, then counting events in EL3 is further controlled by PMEVTYPER<n>_EL0.M.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

EL0 filtering. Controls counting events in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of events.
0b1	The PE does not count events in EL0.

If Secure and Non-secure states are implemented, then counting events in Non-secure EL0 is further controlled by PMEVTYPER<n>_EL0.NSU.

If FEAT_RME is implemented, then counting events in Realm EL0 is further controlled by PMEVTYPER<n>_EL0.RLU.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting events in Non-secure EL1. If PMEVTYPER<n>_EL0.NSK is not equal to PMEVTYPER<n>_EL0.P, then the PE does not count events in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of events in Non-secure EL1.

NSK	Meaning
0b0	When PMEVTYPER<n>_EL0.P == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.P == 1, the PE does not count events in Non-secure EL1.
0b1	When PMEVTYPER<n>_EL0.P == 0, the PE does not count events in Non-secure EL1. When PMEVTYPER<n>_EL0.P == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 filtering. Controls counting events in Non-secure EL0. If PMEVTYPER<n>_EL0.NSU is not equal to PMEVTYPER<n>_EL0.U, then the PE does not count events in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of events in Non-secure EL0.

NSU	Meaning
0b0	When PMEVTYPER<n>_EL0.U == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.U == 1, the PE does not count events in Non-secure EL0.
0b1	When PMEVTYPER<n>_EL0.U == 0, the PE does not count events in Non-secure EL0. When PMEVTYPER<n>_EL0.U == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 filtering. Controls counting events in EL2.

NSH	Meaning
0b0	The PE does not count events in EL2.
0b1	This mechanism has no effect on filtering of events.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting events in Secure EL2 is further controlled by PMEVTYPER<n>_EL0.SH.

If FEAT_RME is implemented, then counting events in Realm EL2 is further controlled by PMEVTYPER<n>_EL0.RLH.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]

When EL3 is implemented and FEAT_AA64 is implemented:

EL3 filtering. Controls counting events in EL3. If PMEVTYPER<n>_EL0.M is not equal to PMEVTYPER<n>_EL0.P, then the PE does not count events in EL3. Otherwise, this mechanism has no effect on filtering of events in EL3.

M	Meaning
0b0	When PMEVTYPER<n>_EL0.P == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.P == 1, the PE does not count events in EL3.
0b1	When PMEVTYPER<n>_EL0.P == 0, the PE does not count events in EL3. When PMEVTYPER<n>_EL0.P == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MT, bit [25]
When FEAT_MTPMU is implemented or an IMPLEMENTATION DEFINED multi-threaded PMU extension is implemented:

Multithreading.

MT	Meaning
0b0	Count events only on controlling PE.
0b1	Count events from any PE with the same affinity at level 1 and above as this PE.

Unless otherwise stated:

- If the event counts PE cycles when a stall condition is true, then the counter counts Processor cycles when the stall condition is true for all of these PEs.
- If the event counts PE cycles when any other condition is true, then the counter counts Processor cycles when the condition is true for any of these PEs.
- Otherwise, the event counts by the sum of the count across all of these PEs. See 'Multithreaded implementations' and 'Cycle event counting in multithreaded implementations'.

From Armv8.6, the IMPLEMENTATION DEFINED multi-threaded PMU extension is not permitted, meaning if FEAT_MTPMU is not implemented, this field is RES0. See ID_AA64DFR0_EL1.MTPMU.

This field is ignored by the PE and treated as zero when FEAT_MTPMU is implemented and disabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SH, bit [24]
When EL3 is implemented and FEAT_SEL2 is implemented:

Secure EL2 filtering. Controls counting events in Secure EL2. If PMEVTYPER<n>_EL0.SH is equal to PMEVTYPER<n>_EL0.NSH, then the PE does not count events in Secure EL2. Otherwise, this mechanism has no effect on filtering of events in Secure EL2.

SH	Meaning
0b0	When PMEVTYPER<n>_EL0.NSH == 0, the PE does not count events in Secure EL2. When PMEVTYPER<n>_EL0.NSH == 1, this mechanism has no effect on filtering of events.
0b1	When PMEVTYPER<n>_EL0.NSH == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.NSH == 1, the PE does not count events in Secure EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When Secure EL2 is not implemented, access to this field is **RES0**.

Otherwise:

Reserved, RES0.

T, bit [23]

When FEAT_TME is implemented:

Non-Transactional state filtering field. Controls counting of events in Non-transactional state.

T	Meaning
0b0	This field has no effect on the filtering of events.
0b1	Do not count Attributable events in Non-transactional state.

For each Unattributable event, it is IMPLEMENTATION DEFINED whether the filtering applies.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLK, bit [22]

When FEAT_RME is implemented:

Realm EL1 filtering. Controls counting events in Realm EL1. If PMEVTYPER<n>_EL0.RLK is not equal to PMEVTYPER<n>_EL0.P, then the PE does not count events in Realm EL1. Otherwise, this mechanism has no effect on filtering of events in Realm EL1.

RLK	Meaning
0b0	When PMEVTYPER<n>_EL0.P == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.P == 1, the PE does not count events in Realm EL1.
0b1	When PMEVTYPER<n>_EL0.P == 0, the PE does not count events in Realm EL1. When PMEVTYPER<n>_EL0.P == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting events in Realm EL0. If PMEVTYPER<n>_EL0.RLU is not equal to PMEVTYPER<n>_EL0.U, then the PE does not count events in Realm EL0. Otherwise, this mechanism has no effect on filtering of events in Realm EL0.

RLU	Meaning
0b0	When PMEVTYPER<n>_EL0.U == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.U == 1, the PE does not count events in Realm EL0.
0b1	When PMEVTYPER<n>_EL0.U == 0, the PE does not count events in Realm EL0. When PMEVTYPER<n>_EL0.U == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]

When FEAT_RME is implemented:

Realm EL2 filtering. Controls counting events in Realm EL2. If PMEVTYPER<n>_EL0.RLH is equal to PMEVTYPER<n>_EL0.NSH, then the PE does not count events in Realm EL2. Otherwise, this mechanism has no effect on filtering of events in Realm EL2.

RLH	Meaning
0b0	When PMEVTYPER<n>_EL0.NSH == 0, the PE does not count events in Realm EL2. When PMEVTYPER<n>_EL0.NSH == 1, this mechanism has no effect on filtering of events.
0b1	When PMEVTYPER<n>_EL0.NSH == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.NSH == 1, the PE does not count events in Realm EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:16]

Reserved, RES0.

evtCount[15:10], bits [15:10]

When FEAT_PMUv3p1 is implemented:

Extension to evtCount[9:0]. For more information, see evtCount[9:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

evtCount[9:0], bits [9:0]

Event to count.

The event number of the event that is counted by event counter PMU.PMEVCNTR<n>_EL0.

The ranges of event numbers allocated to each type of event are shown in 'Allocation of the PMU event number space'.

If FEAT_PMUv3p8 is implemented and PMEVTYPER<n>_EL0.evtCount is programmed to an event that is reserved or not supported by the PE, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.

Note

Arm recommends this behavior for all implementations of FEAT_PMUv3.

Otherwise, if PMEVTYPER<n>_EL0.evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range `0x0000` to `0x003F`, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.
- If FEAT_PMUv3p1 is implemented, for the range `0x4000` to `0x403F`, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.
- For other values, it is UNPREDICTABLE what event, if any, is counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is UNKNOWN.

Note

UNPREDICTABLE means the event must not expose privileged information.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMEVTYPER<n>_EL0

If FEAT_PMUv3_EXT32 is implemented and any of the following apply, then bits [63:32] of this register are accessible at offset `0xA00` + (4*n):

- FEAT_PMUv3_TH is implemented.
- FEAT_PMUv3p8 is implemented.
- FEAT_PMUv3_SME is implemented.

Otherwise accesses at this offset are IMPLEMENTATION DEFINED.

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented

[63:0] Accessible at offset `0x400 + (8 * n)` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.

- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXTPMN is implemented, IsRange3Counter(n), and !IsMostSecureAccess(addrdesc), accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented

[31:0] Accessible at offset `0x400 + (4 * n)` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXTPMN is implemented, IsRange3Counter(n), and !IsMostSecureAccess(addrdesc), accesses to this register are **RAZ/WI**.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented and (FEAT_PMUv3_TH is implemented, or FEAT_PMUv3p8 is implemented, or FEAT_PMUv3_SME is implemented)

[63:32] Accessible at offset `0xA00 + (4 * n)` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXTPMN is implemented, IsRange3Counter(n), and !IsMostSecureAccess(addrdesc), accesses to this register are **RAZ/WI**.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMICFILTR_EL0, Performance Monitors Instruction Counter Filter Register

The PMICFILTR_EL0 characteristics are:

Purpose

Configures the Instruction Counter.

Configuration

External register PMICFILTR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMICFILTR_EL0[63:0].

This register is present only when FEAT_PMUv3_ICNTR is implemented. Otherwise, direct accesses to PMICFILTR_EL0 are RES0.

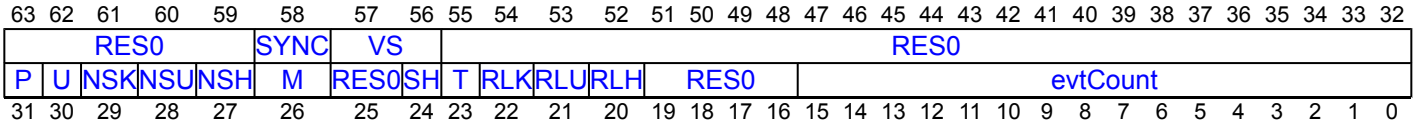
PMICFILTR_EL0 is in the Core power domain.

Attributes

PMICFILTR_EL0 is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:59]

Reserved, RES0.

SYNC, bit [58]

When FEAT_SEBEP is implemented:

Synchronous mode. Controls whether a PMU Profiling exception generated by the counter is synchronous or asynchronous.

SYNC	Meaning
0b0	Asynchronous PMU Profiling exception is enabled.
0b1	Synchronous PMU Profiling exception is enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VS, bits [57:56]

When FEAT_PMUv3_SME is implemented:

SVE mode filtering. Controls counting instructions in Streaming and Non-streaming SVE modes.

VS	Meaning
0b00	This mechanism has no effect on the filtering of instructions.
0b01	The PE does not count instructions in Streaming SVE mode.
0b10	The PE does not count instructions in Non-streaming SVE mode.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [55:32]

Reserved, RES0.

P, bit [31]

EL1 filtering. Controls counting instructions in EL1.

P	Meaning
0b0	This mechanism has no effect on filtering of instructions.
0b1	The PE does not count instructions in EL1.

If Secure and Non-secure states are implemented, then counting instructions in Non-secure EL1 is further controlled by PMICFILTR_EL0.NSK.

If FEAT_RME is implemented, then counting instructions in Realm EL1 is further controlled by PMICFILTR_EL0.RLK.

If EL3 is implemented, then counting instructions in EL3 is further controlled by PMICFILTR_EL0.M.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

EL0 filtering. Controls counting instructions in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of instructions.
0b1	The PE does not count instructions in EL0.

If Secure and Non-secure states are implemented, then counting instructions in Non-secure EL0 is further controlled by PMICFILTR_EL0.NSU.

If FEAT_RME is implemented, then counting instructions in Realm EL0 is further controlled by PMICFILTR_EL0.RLU.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting instructions in Non-secure EL1. If PMICFILTR_EL0.NSK is not equal to PMICFILTR_EL0.P, then the PE does not count instructions in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of instructions in Non-secure EL1.

NSK	Meaning
0b0	When PMICFILTR_EL0.P == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.P == 1, the PE does not count instructions in Non-secure EL1.
0b1	When PMICFILTR_EL0.P == 0, the PE does not count instructions in Non-secure EL1. When PMICFILTR_EL0.P == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 filtering. Controls counting instructions in Non-secure EL0. If PMICFILTR_EL0.NSU is not equal to PMICFILTR_EL0.U, then the PE does not count instructions in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of instructions in Non-secure EL0.

NSU	Meaning
0b0	When PMICFILTR_EL0.U == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.U == 1, the PE does not count instructions in Non-secure EL0.
0b1	When PMICFILTR_EL0.U == 0, the PE does not count instructions in Non-secure EL0. When PMICFILTR_EL0.U == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 filtering. Controls counting instructions in EL2.

NSH	Meaning
0b0	The PE does not count instructions in EL2.
0b1	This mechanism has no effect on filtering of instructions.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting instructions in Secure EL2 is further controlled by PMICFILTR_EL0.SH.

If FEAT_RME is implemented, then counting instructions in Realm EL2 is further controlled by PMICFILTR_EL0.RLH.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]

When EL3 is implemented:

EL3 filtering. Controls counting instructions in EL3. If PMICFILTR_EL0.M is not equal to PMICFILTR_EL0.P, then the PE does not count instructions in EL3. Otherwise, this mechanism has no effect on filtering of instructions in EL3.

M	Meaning
0b0	When PMICFILTR_EL0.P == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.P == 1, the PE does not count instructions in EL3.
0b1	When PMICFILTR_EL0.P == 0, the PE does not count instructions in EL3. When PMICFILTR_EL0.P == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [25]

Reserved, RES0.

SH, bit [24]

When EL3 is implemented and FEAT_SEL2 is implemented:

Secure EL2 filtering. Controls counting instructions in Secure EL2. If PMICFILTR_EL0.SH is equal to PMICFILTR_EL0.NSH, then the PE does not count instructions in Secure EL2. Otherwise, this mechanism has no effect on filtering of instructions in Secure EL2.

SH	Meaning
0b0	When PMICFILTR_EL0.NSH == 0, the PE does not count instructions in Secure EL2. When PMICFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of instructions.
0b1	When PMICFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.NSH == 1, the PE does not count instructions in Secure EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When Secure EL2 is not implemented, access to this field is **RES0**.

Otherwise:

Reserved, RES0.

T, bit [23]
When FEAT_TME is implemented:

Non-Transactional state filtering field. Controls counting of instructions in Non-transactional state.

T	Meaning
0b0	This field has no effect on the filtering of instructions.
0b1	Do not count Attributable instructions in Non-transactional state.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLK, bit [22]
When FEAT_RME is implemented:

Realm EL1 filtering. Controls counting instructions in Realm EL1. If PMICFILTR_EL0.RLK is not equal to PMICFILTR_EL0.P, then the PE does not count instructions in Realm EL1. Otherwise, this mechanism has no effect on filtering of instructions in Realm EL1.

RLK	Meaning
0b0	When PMICFILTR_EL0.P == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.P == 1, the PE does not count instructions in Realm EL1.
0b1	When PMICFILTR_EL0.P == 0, the PE does not count instructions in Realm EL1. When PMICFILTR_EL0.P == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]
When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting instructions in Realm EL0. If PMICFILTR_EL0.RLU is not equal to PMICFILTR_EL0.U, then the PE does not count instructions in Realm EL0. Otherwise, this mechanism has no effect on filtering of instructions in Realm EL0.

RLU	Meaning
0b0	When PMICFILTR_EL0.U == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.U == 1, the PE does not count instructions in Realm EL0.
0b1	When PMICFILTR_EL0.U == 0, the PE does not count instructions in Realm EL0. When PMICFILTR_EL0.U == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]

When FEAT_RME is implemented:

Realm EL2 filtering. Controls counting instructions in Realm EL2. If PMICFILTR_EL0.RLH is equal to PMICFILTR_EL0.NSH, then the PE does not count instructions in Realm EL2. Otherwise, this mechanism has no effect on filtering of instructions in Realm EL2.

RLH	Meaning
0b0	When PMICFILTR_EL0.NSH == 0, the PE does not count instructions in Realm EL2. When PMICFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of instructions.
0b1	When PMICFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.NSH == 1, the PE does not count instructions in Realm EL2.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count.

Reads as 0x0008.

Access to this field is **RO**.

Accessing PMICFILTR_EL0

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT32 is implemented

[31:0] Accessible at offset `0x480` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT64 is implemented

Accessible at offset `0x500` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented

[63:32] Accessible at offset `0xA80` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMICNTR_EL0, Performance Monitors Instruction Counter Register

The PMICNTR_EL0 characteristics are:

Purpose

If event counting is not prohibited and the instruction counter is enabled, the counter increments for each architecturally-executed instruction, according to the configuration specified by PMICFILTR_EL0.

Configuration

External register PMICNTR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMICNTR_EL0[63:0].

This register is present only when FEAT_PMUv3_ICNTR is implemented. Otherwise, direct accesses to PMICNTR_EL0 are RES0.

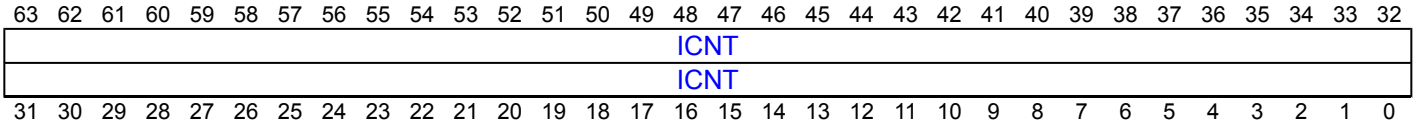
PMICNTR_EL0 is in the Core power domain.

Attributes

PMICNTR_EL0 is a 64-bit register.

This register is part of the PMU block.

Field descriptions



ICNT, bits [63:0]

Instruction Counter.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMICNTR_EL0

Accesses to this register use the following encodings:

Accessible at offset `0x100` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMICNTSVR_EL1, Performance Monitors Instruction Count Saved Value Register

The PMICNTSVR_EL1 characteristics are:

Purpose

Captures the PMU Instruction counter, PMICNTR_EL0.

Configuration

External register PMICNTSVR_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMICNTSVR_EL1[63:0].

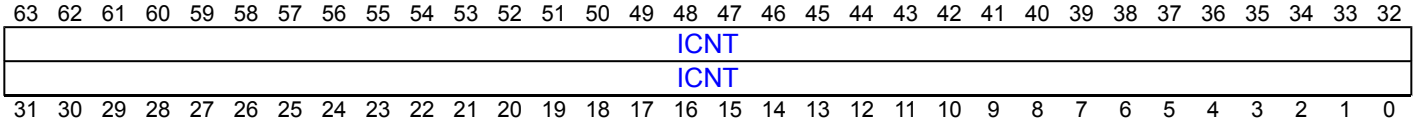
This register is present only when FEAT_PMUv3_ICNTR is implemented and FEAT_PMUv3_SS is implemented. Otherwise, direct accesses to PMICNTSVR_EL1 are RES0.

Attributes

PMICNTSVR_EL1 is a 64-bit register.

This register is part of the PMU block.

Field descriptions



ICNT, bits [63:0]

Sampled Instruction Count. The value of PMICNTR_EL0 at the last successful Capture event.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMICNTSVR_EL1

Accesses to this register use the following encodings:

Accessible at offset `0x700` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMSSAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMIIDR, Performance Monitors Implementation Identification Register

The PMIIDR characteristics are:

Purpose

Provides discovery information about the Performance Monitor component.

Configuration

This register is present only when (FEAT_PMUv3_EXT32 is implemented and an implementation implements PMIIDR) or FEAT_PMUv3_EXT64 is implemented. Otherwise, direct accesses to PMIIDR are RES0.

Attributes

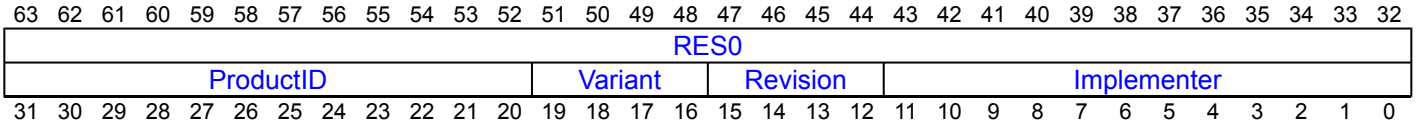
PMIIDR is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented:



Bits [63:32]

Reserved, RES0.

ProductID, bits [31:20]

Part number, bits [11:0]. The part number is selected by the designer of the component.

Matches the PMPIDR1.PART_1, PMPIDR0.PART_0 fields if present.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [19:16]

Component major revision.

Defines either a variant of the component defined by PMIIDR.ProductID, or the major revision of the component.

When defining a major revision, PMIIDR.Variant and PMIIDR.Revision together form the revision number of the component, with this field being the most significant part.

When a component is changed, PMIIDR.Variant or PMIIDR.Revision is increased to ensure that software can differentiate between different revisions of the component. If this field is increased, PMIIDR.Revision should be set to `0b0000`.

Matches the PMPIDR2.REVISION field, if present.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Component minor revision.

PMIIDR.Variant and PMIIDR.Revision together form the revision number of the component, with this field being the least significant part.

When a component is changed, PMIIDR.Variant or PMIIDR.Revision is increased to ensure that software can differentiate between different revisions of the component. If PMIIDR.Variant field is increased, this field should be set to `0b0000`, otherwise the value in this field should be increased.

Matches the PMPIDR3.REVAND field, if present.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the PMU.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for PMIIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

If PMPIDR4 is implemented, PMPIDR4.DES_2 matches bits [11:8] of this field.

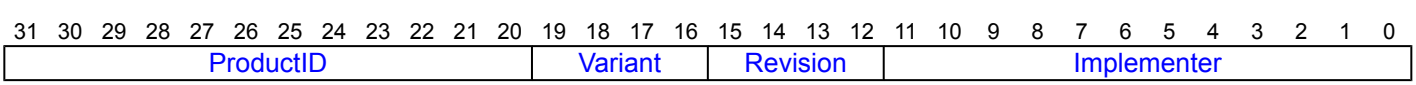
If PMPIDR2 is implemented, PMPIDR2.DES_1 matches bits [6:4] of this field.

If PMPIDR1 is implemented, PMPIDR1.DES_0 matches bits [3:0] of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:



ProductID, bits [31:20]

Part number, bits [11:0]. The part number is selected by the designer of the component.

Matches the PMPIDR1.PART_1, PMPIDR0.PART_0 fields if present.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [19:16]

Component major revision.

Defines either a variant of the component defined by PMIIDR.ProductID, or the major revision of the component.

When defining a major revision, PMIIDR.Variant and PMIIDR.Revision together form the revision number of the component, with this field being the most significant part.

When a component is changed, PMIIDR.Variant or PMIIDR.Revision is increased to ensure that software can differentiate between different revisions of the component. If this field is increased, PMIIDR.Revision should be set to `0b0000`.

Matches the PMPIDR2.REVISION field, if present.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Component minor revision.

PMIIDR.Variant and PMIIDR.Revision together form the revision number of the component, with this field being the least significant part.

When a component is changed, PMIIDR.Variant or PMIIDR.Revision is increased to ensure that software can differentiate between different revisions of the component. If PMIIDR.Variant field is increased, this field should be set to `0b0000`, otherwise the value in this field should be increased.

Matches the PMPIDR3.REVAND field, if present.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the PMU.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for PMIIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

If PMPIDR4 is implemented, PMPIDR4.DES_2 matches bits [11:8] of this field.

If PMPIDR2 is implemented, PMPIDR2.DES_1 matches bits [6:4] of this field.

If PMPIDR1 is implemented, PMPIDR1.DES_0 matches bits [3:0] of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMIIDR

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT is implemented

Accessible at offset `0xE08` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTEN, Performance Monitors Interrupt Enable register

The PMINTEN characteristics are:

Purpose

Enables the generation of interrupt requests on overflows from the Cycle Count Register, PMCCNTR_EL0, and the event counters PMEVCNTR<n>_EL0.

Configuration

External register PMINTEN bits [63:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[63:0].

External register PMINTEN bits [63:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[63:0].

External register PMINTEN bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0].

External register PMINTEN bits [31:0] are architecturally mapped to AArch32 System register PMINTENSET[31:0].

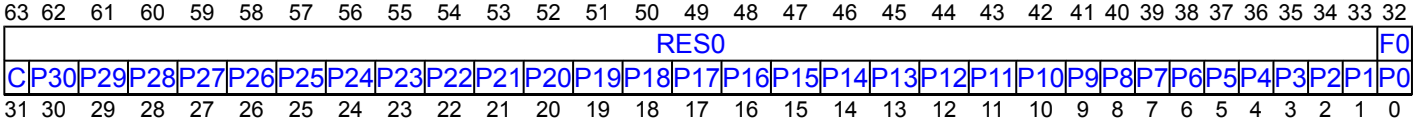
This register is present only when FEAT_PMUv3_EXT64 is implemented. Otherwise, direct accesses to PMINTEN are RES0.

Attributes

PMINTEN is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Interrupt request on unsigned overflow of PMICNTR_EL0 enable.

F0	Meaning
0b0	Interrupt request on unsigned overflow of PMICNTR_EL0 disabled.
0b1	Interrupt request on unsigned overflow of PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 unsigned overflow interrupt request enable bit. Possible values are:

C	Meaning
0b0	The cycle counter overflow interrupt request is disabled.
0b1	The cycle counter overflow interrupt request is enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

P<m>, bit [m], for m = 30 to 0

Event counter unsigned overflow interrupt request enable bit for PMEVCNTR<m>_EL0.

P<m>	Meaning
0b0	The PMEVCNTR<m>_EL0 event counter interrupt request is disabled.
0b1	The PMEVCNTR<m>_EL0 event counter interrupt request is enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).
  - !IsMostSecureAccess(addrdesc).
- Otherwise, access to this field is **RW**.

Accessing PMINTEN

Accesses to this register use the following encodings:

Accessible at offset 0xC50 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENCLR_EL1, Performance Monitors Interrupt Enable Clear Register

The PMINTENCLR_EL1 characteristics are:

Purpose

Allows software to disable the generation of interrupt requests or, when FEAT_EBEP is implemented, PMU Profiling exceptions on overflows from the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which overflow interrupt requests or PMU Profiling exceptions are enabled.

Configuration

External register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[31:0] when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented.

External register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[31:0] when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented.

External register PMINTENCLR_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[63:0] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMINTENCLR_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[63:0] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0].

External register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENSET[31:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMINTENCLR_EL1 are RES0.

PMINTENCLR_EL1 is in the Core power domain.

Attributes

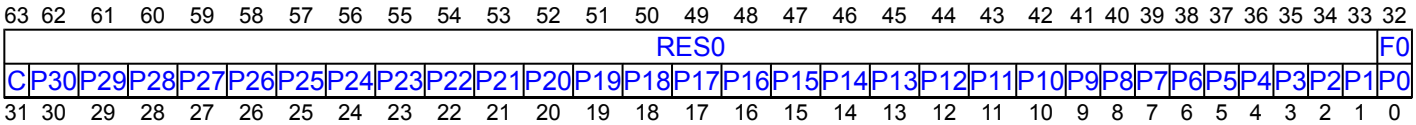
PMINTENCLR_EL1 is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented:



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enable status.

F0	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

C, bit [31]

Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable status.

C	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).
  - !IsMostSecureAccess(addrdesc).
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

C, bit [31]

Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable status.

C	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).
  - !IsMostSecureAccess(addrdesc).
- When SoftwareLockStatus(), access to this field is **RO**.

- Otherwise, access to this field is **W1C**.

Accessing PMINTENCLR_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented

[63:0] Accessible at offset `0xC60` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXT32 is implemented and SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented

[31:0] Accessible at offset `0xC60` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENSET_EL1, Performance Monitors Interrupt Enable Set Register

The PMINTENSET_EL1 characteristics are:

Purpose

Allows software to enable the generation of interrupt requests or, when FEAT_EBEP is implemented, PMU Profiling exceptions on overflows from the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which overflow interrupt requests or PMU Profiling exceptions are enabled.

Configuration

External register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[31:0] when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented.

External register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[31:0] when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented.

External register PMINTENSET_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[63:0] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMINTENSET_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[63:0] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENSET[31:0].

External register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMINTENSET_EL1 are RES0.

PMINTENSET_EL1 is in the Core power domain.

Attributes

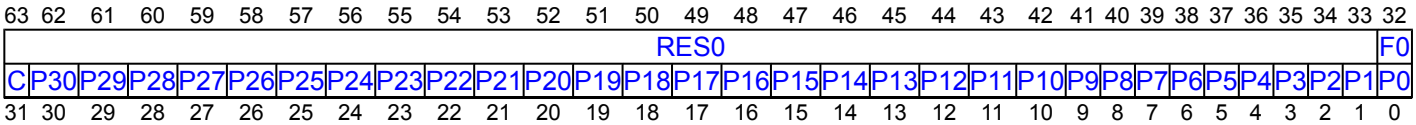
PMINTENSET_EL1 is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented:



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enable status.

F0	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

Otherwise:

Reserved, RES0.

C, bit [31]

Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable status.

C	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).
  - !IsMostSecureAccess(addrdesc).
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

C, bit [31]

Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable status.

C	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).
  - !IsMostSecureAccess(addrdesc).
- When SoftwareLockStatus(), access to this field is **RO**.

- Otherwise, access to this field is **W1S**.

Accessing PMINTENSET_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented

[63:0] Accessible at offset `0xC40` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXT32 is implemented and SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented

[31:0] Accessible at offset `0xC40` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMITCTRL, Performance Monitors Integration mode Control register

The PMITCTRL characteristics are:

Purpose

Enables the Performance Monitors to switch from default mode into integration mode, where test software can control directly the inputs and outputs of the PE, for integration testing or topology detection.

Configuration

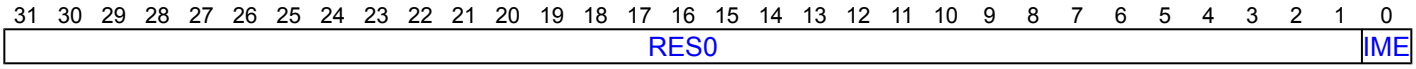
This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMITCTRL. Otherwise, direct accesses to PMITCTRL are RES0.

Attributes

PMITCTRL is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:1]

Reserved, RES0.

IME, bit [0]

Integration mode enable. When IME == 1, the device reverts to an integration mode to enable integration testing or topology detection.

IME	Meaning
0b0	Normal operation.
0b1	Integration mode enabled.

The integration mode behavior is IMPLEMENTATION DEFINED.

The following resets apply:

- If the register is implemented in the Core power domain:
  - On a Cold reset, this field resets to 0.
  - On an External debug reset, the value of this field is unchanged.
  - On a Warm reset, the value of this field is unchanged.
- If the register is implemented in the External debug power domain:
  - On a Cold reset, the value of this field is unchanged.
  - On an External debug reset, this field resets to 0.
  - On a Warm reset, the value of this field is unchanged.

Accessing PMITCTRL

Accesses to this register use the following encodings:

Accessible at offset `0xF00` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMLAR, Performance Monitors Lock Access Register

The PMLAR characteristics are:

Purpose

Allows or disallows access to the Performance Monitors registers through a memory-mapped interface.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Performance Monitors registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Performance Monitors registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMLAR are RES0.

If FEAT_DoPD is implemented, Software Lock is not implemented by the architecturally-defined debug components of the PE in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Software uses PMLAR to set or clear the lock, and PMLSR to check the current status of the lock.

Attributes

PMLAR is a 32-bit register.

This register is part of the PMU block.

Field descriptions

When PMU Software Lock is implemented:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KEY																															

KEY, bits [31:0]

Lock Access control. Writing the key value `0xC5ACCE55` to this field unlocks the lock, enabling write accesses to this component's registers through a memory-mapped interface.

Writing any other value to this register locks the lock, disabling write accesses to this component's registers through a memory mapped interface.

Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0																															

Otherwise

Bits [31:0]

Reserved, RES0.

Accessing PMLAR

Accesses to this register use the following encodings:

Accessible at offset `0xFB0` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMLSR, Performance Monitors Lock Status Register

The PMLSR characteristics are:

Purpose

Indicates the current status of the software lock for Performance Monitors registers.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Performance Monitors registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Performance Monitors registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMLSR are RES0.

If FEAT_DoPD is implemented, Software Lock is not implemented by the architecturally-defined debug components of the PE in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

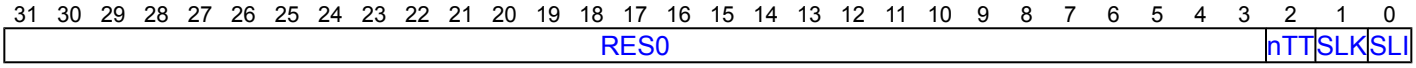
Software uses PMLAR to set or clear the lock, and PMLSR to check the current status of the lock.

Attributes

PMLSR is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:3]

Reserved, RES0.

nTT, bit [2]

Not thirty-two bit access required.

Reads as `0b0`.

Access to this field is **RO**.

SLK, bit [1]

When PMU Software Lock is implemented and FEAT_DoPD is not implemented:

Software Lock status for this component. For an access to LSR that is not a memory-mapped access, or when Software Lock is not implemented, this field is RES0.

For memory-mapped accesses when Software Lock is implemented, possible values of this field are:

SLK	Meaning
`0b0`	Lock clear. Writes are permitted to this component's registers.
`0b1`	Lock set. Writes to this component's registers are ignored, and reads have no side effects.

The reset behavior of this field is:

- On an External debug reset, this field resets to `'1'`.

Otherwise:

Reserved, RAZ.

SLI, bit [0]

Software Lock implemented. For an access to LSR that is not a memory-mapped access, this field is RAZ.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SLI	Meaning
`0b0`	Software Lock not implemented or not memory-mapped access.
`0b1`	Software Lock implemented and memory-mapped access.

Access to this field is **RO**.

Accessing PMLSR

Accesses to this register use the following encodings:

Accessible at offset `0xFB4` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMMIR, Performance Monitors Machine Identification Register

The PMMIR characteristics are:

Purpose

Describes Performance Monitors parameters specific to the implementation.

Configuration

External register PMMIR bits [31:0] are architecturally mapped to AArch32 System register PMMIR[31:0].

External register PMMIR bits [31:0] are architecturally mapped to AArch64 System register PMMIR_EL1[31:0].

External register PMMIR bits [63:0] are architecturally mapped to AArch64 System register PMMIR_EL1[63:0] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

This register is present only when FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p4 is implemented. Otherwise, direct accesses to PMMIR are RES0.

PMMIR is in the Core power domain.

Attributes

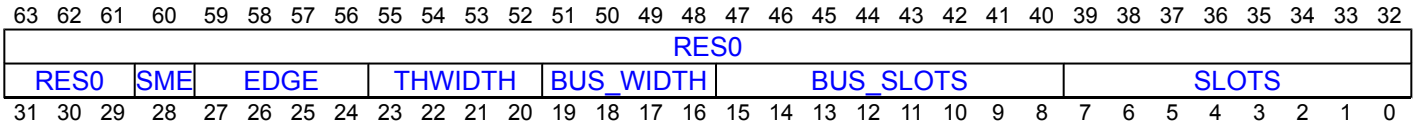
PMMIR is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented:



Bits [63:29]

Reserved, RES0.

SME, bit [28]

PMUv3 for SME. Adds support for the Streaming SVE mode filter.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SME	Meaning
0b0	Streaming SVE mode filter not implemented.
0b1	Adds support for the Streaming SVE mode filter.

All other values are reserved.

FEAT_PMUv3_SME implements the functionality identified by the value 1.

Access to this field is **RO**.

EDGE, bits [27:24]

PMU event edge detection. With PMMIR_EL1.THWIDTH, indicates implementation of event counter thresholding features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EDGE	Meaning
0b0000	FEAT_PMUv3_EDGE is not implemented.
0b0001	FEAT_PMUv3_EDGE is implemented.
0b0010	As 0b0001, and adds support for threshold value linking between a pair of counters.

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, the only permitted value is 0b0000.

FEAT_PMUv3_EDGE implements the functionality identified by the value 0b0001.

FEAT_PMUv3_TH2 implements the functionality identified by the value 0b0010.

Access to this field is **RO**.

THWIDTH, bits [23:20]

PMEVTYPER<n>_EL0.TH width. Indicates implementation of the FEAT_PMUv3_TH feature, and, if implemented, the size of the PMEVTYPER<n>_EL0.TH field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

THWIDTH	Meaning
0b0000	FEAT_PMUv3_TH is not implemented.
0b0001	1 bit. PMEVTYPER<n>_EL0.TH[11:1] are RES0.
0b0010	2 bits. PMEVTYPER<n>_EL0.TH[11:2] are RES0.
0b0011	3 bits. PMEVTYPER<n>_EL0.TH[11:3] are RES0.
0b0100	4 bits. PMEVTYPER<n>_EL0.TH[11:4] are RES0.
0b0101	5 bits. PMEVTYPER<n>_EL0.TH[11:5] are RES0.
0b0110	6 bits. PMEVTYPER<n>_EL0.TH[11:6] are RES0.
0b0111	7 bits. PMEVTYPER<n>_EL0.TH[11:7] are RES0.
0b1000	8 bits. PMEVTYPER<n>_EL0.TH[11:8] are RES0.
0b1001	9 bits. PMEVTYPER<n>_EL0.TH[11:9] are RES0.
0b1010	10 bits. PMEVTYPER<n>_EL0.TH[11:10] are RES0.
0b1011	11 bits. PMEVTYPER<n>_EL0.TH[11] is RES0.
0b1100	12 bits.

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, this field is zero.

Otherwise, the largest value that can be written to PMEVTYPER<n>_EL0.TH is $2^{(\text{PMMIR.THWIDTH})}$ minus one.

Access to this field is **RO**.

BUS_WIDTH, bits [19:16]

Bus width. Indicates the number of bytes each BUS_ACCESS event relates to. Encoded as Log_2(number of bytes), plus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BUS_WIDTH	Meaning
0b0000	The information is not available.
0b0011	Four bytes.
0b0100	8 bytes.
0b0101	16 bytes.
0b0110	32 bytes.
0b0111	64 bytes.
0b1000	128 bytes.
0b1001	256 bytes.
0b1010	512 bytes.
0b1011	1024 bytes.
0b1100	2048 bytes.

All other values are reserved.

Each transfer is up to this number of bytes. An access might be smaller than the bus width.

When this field is nonzero, each access counted by BUS_ACCESS is at most BUS_WIDTH bytes. An implementation might treat a wide bus as multiple narrower buses, such that a wide access on the bus increments the BUS_ACCESS counter by more than one.

Access to this field is **RO**.

BUS_SLOTS, bits [15:8]

Bus count. The largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle.

When this field is nonzero, the largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle is BUS_SLOTS.

If the bus count information is not available, this field will read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

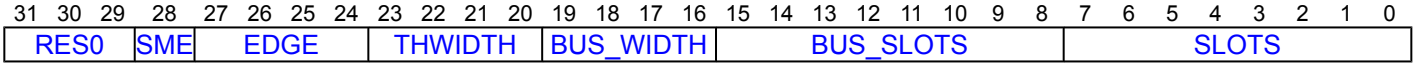
SLOTS, bits [7:0]

Operation width. The largest value by which the STALL_SLOT event might increment in a single cycle. If the STALL_SLOT event is not implemented, this field might read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:



Bits [31:29]

Reserved, RES0.

SME, bit [28]

PMUv3 for SME. Adds support for the Streaming SVE mode filter.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SME	Meaning
0b0	Streaming SVE mode filter not implemented.
0b1	Adds support for the Streaming SVE mode filter.

All other values are reserved.

FEAT_PMUv3_SME implements the functionality identified by the value 1.

Access to this field is **RO**.

EDGE, bits [27:24]

PMU event edge detection. With PMMIR_EL1.THWIDTH, indicates implementation of event counter thresholding features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EDGE	Meaning
0b0000	FEAT_PMUv3_EDGE is not implemented.
0b0001	FEAT_PMUv3_EDGE is implemented.
0b0010	As 0b0001, and adds support for threshold value linking between a pair of counters.

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, the only permitted value is 0b0000.

FEAT_PMUv3_EDGE implements the functionality identified by the value 0b0001.

FEAT_PMUv3_TH2 implements the functionality identified by the value 0b0010.

Access to this field is **RO**.

THWIDTH, bits [23:20]

PMEVTYPER<n>_EL0.TH width. Indicates implementation of the FEAT_PMUv3_TH feature, and, if implemented, the size of the PMEVTYPER<n>_EL0.TH field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

THWIDTH	Meaning
0b0000	FEAT_PMUv3_TH is not implemented.
0b0001	1 bit. PMEVTYPER<n>_EL0.TH[11:1] are RES0.
0b0010	2 bits. PMEVTYPER<n>_EL0.TH[11:2] are RES0.
0b0011	3 bits. PMEVTYPER<n>_EL0.TH[11:3] are RES0.
0b0100	4 bits. PMEVTYPER<n>_EL0.TH[11:4] are RES0.
0b0101	5 bits. PMEVTYPER<n>_EL0.TH[11:5] are RES0.
0b0110	6 bits. PMEVTYPER<n>_EL0.TH[11:6] are RES0.
0b0111	7 bits. PMEVTYPER<n>_EL0.TH[11:7] are RES0.
0b1000	8 bits. PMEVTYPER<n>_EL0.TH[11:8] are RES0.
0b1001	9 bits. PMEVTYPER<n>_EL0.TH[11:9] are RES0.
0b1010	10 bits. PMEVTYPER<n>_EL0.TH[11:10] are RES0.
0b1011	11 bits. PMEVTYPER<n>_EL0.TH[11] is RES0.
0b1100	12 bits.

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, this field is zero.

Otherwise, the largest value that can be written to PMEVTYPER<n>_EL0.TH is $2^{(\text{PMMIR.THWIDTH})}$ minus one.

Access to this field is **RO**.

BUS_WIDTH, bits [19:16]

Bus width. Indicates the number of bytes each BUS_ACCESS event relates to. Encoded as Log_2(number of bytes), plus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BUS_WIDTH	Meaning
0b0000	The information is not available.
0b0011	Four bytes.
0b0100	8 bytes.
0b0101	16 bytes.
0b0110	32 bytes.
0b0111	64 bytes.
0b1000	128 bytes.
0b1001	256 bytes.
0b1010	512 bytes.
0b1011	1024 bytes.
0b1100	2048 bytes.

All other values are reserved.

Each transfer is up to this number of bytes. An access might be smaller than the bus width.

When this field is nonzero, each access counted by BUS_ACCESS is at most BUS_WIDTH bytes. An implementation might treat a wide bus as multiple narrower buses, such that a wide access on the bus increments the BUS_ACCESS counter by more than one.

Access to this field is **RO**.

BUS_SLOTS, bits [15:8]

Bus count. The largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle.

When this field is nonzero, the largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle is BUS_SLOTS.

If the bus count information is not available, this field will read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SLOTS, bits [7:0]

Operation width. The largest value by which the STALL_SLOT event might increment in a single cycle. If the STALL_SLOT event is not implemented, this field might read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMMIR

If the Core power domain is off or in a low-power state, access to this register returns an Error.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented

[63:0] Accessible at offset 0xE40 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented

[31:0] Accessible at offset 0xE40 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVS, Performance Monitors Overflow Flag Status register

The PMOVS characteristics are:

Purpose

The unsigned overflow flags for the Cycle Count Register, PMCCNTR_EL0, and each of the implemented event counters PMEVCNTR<n>.

Configuration

External register PMOVS bits [63:0] are architecturally mapped to AArch64 System register PMOVSSET_EL0[63:0].

External register PMOVS bits [63:0] are architecturally mapped to AArch64 System register PMOVSCLR_EL0[63:0].

External register PMOVS bits [31:0] are architecturally mapped to AArch32 System register PMOVSR[31:0].

External register PMOVS bits [31:0] are architecturally mapped to AArch32 System register PMOVSSET[31:0].

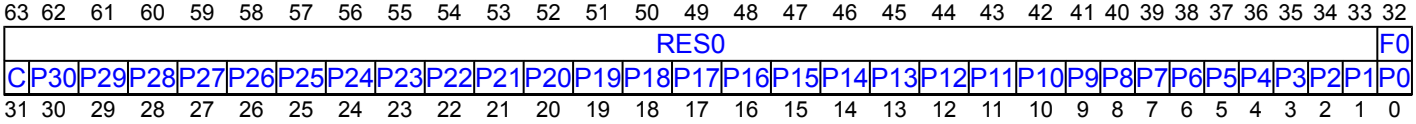
This register is present only when FEAT_PMUv3_EXT64 is implemented. Otherwise, direct accesses to PMOVS are RES0.

Attributes

PMOVS is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

PMICNTR_EL0 unsigned overflow flag.

F0	Meaning
0b0	PMICNTR_EL0 has not overflowed.
0b1	PMICNTR_EL0 has overflowed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

C, bit [31]

Cycle counter unsigned overflow flag.

C	Meaning
0b0	The cycle counter has not overflowed since this bit was last cleared to 0.
0b1	The cycle counter has overflowed since this bit was last cleared to 0.

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

P<m>, bit [m], for m = 30 to 0

Event counter unsigned overflow bit for PMEVCNTR<m>_EL0.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 has not overflowed since this bit was last cleared to 0.
0b1	PMEVCNTR<m>_EL0 has overflowed since this bit was last cleared to 0.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>_EL0[31:0] or unsigned overflow of PMEVCNTR<m>_EL0[63:0].

When FEAT_PMUv3_EXTPMN is implemented, MDCR_EL2.HLP and PMCR_EL0.LP are applicable only for event counters in the second and first range. For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).
  - !IsMostSecureAccess(addrdesc).
- Otherwise, access to this field is **RW**.

Accessing PMOVS

Accesses to this register use the following encodings:

Accessible at offset 0xC90 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVSCLR_EL0, Performance Monitors Overflow Flag Status Clear register

The PMOVSCLR_EL0 characteristics are:

Purpose

Allows software to clear the unsigned overflow flags for the following counters to 0:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows the current unsigned overflow flag values.

Configuration

External register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMOVSSET_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented.

External register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMOVSCLR_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented.

External register PMOVSCLR_EL0 bits [63:32] are architecturally mapped to AArch64 System register PMOVSSET_EL0[63:32] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMOVSCLR_EL0 bits [63:32] are architecturally mapped to AArch64 System register PMOVSCLR_EL0[63:32] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSR[31:0].

External register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSSET[31:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMOVSCLR_EL0 are RES0.

PMOVSCLR_EL0 is in the Core power domain.

Attributes

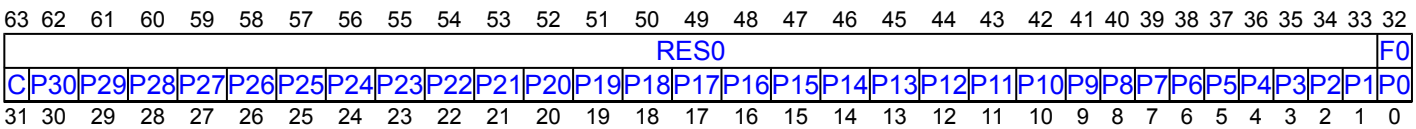
PMOVSCLR_EL0 is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented:



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Unsigned overflow flag for PMICNTR_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMICNTR_EL0 to 0. On reads, returns the unsigned overflow flag for PMICNTR_EL0.

F0	Meaning
0b0	PMICNTR_EL0 has not overflowed.
0b1	PMICNTR_EL0 has overflowed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

C, bit [31]

Unsigned overflow flag for PMCCNTR_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMCCNTR_EL0 to 0. On reads, returns the unsigned overflow flag for PMCCNTR_EL0 overflow status.

C	Meaning
0b0	PMCCNTR_EL0 has not overflowed.
0b1	PMCCNTR_EL0 has overflowed.

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

Unsigned overflow flag for PMEVCNTR<m>_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMEVCNTR<m>_EL0 to 0. On reads, returns the unsigned overflow flag for PMEVCNTR<m>_EL0 overflow status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 has not overflowed.
0b1	PMEVCNTR<m>_EL0 has overflowed.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>_EL0[31:0] or unsigned overflow of PMEVCNTR<m>_EL0[63:0].

When FEAT_PMUv3_EXTPMN is implemented, MDCR_EL2.HLP and PMCR_EL0.LP are applicable only for event counters in the second and first range. For more information about event counter ranges, see MDCR_EL2.HPMN.

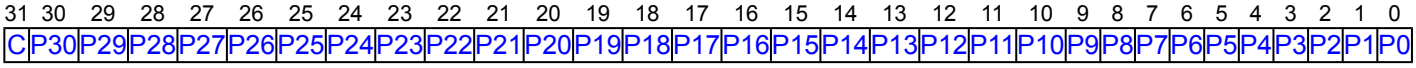
The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).
  - !IsMostSecureAccess(addrdesc).
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

Otherwise:



C, bit [31]

Unsigned overflow flag for PMCCNTR_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMCCNTR_EL0 to 0. On reads, returns the unsigned overflow flag for PMCCNTR_EL0 overflow status.

C	Meaning
0b0	PMCCNTR_EL0 has not overflowed.
0b1	PMCCNTR_EL0 has overflowed.

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1C**.

P<m>, bit [m], for m = 30 to 0

Unsigned overflow flag for PMEVCNTR<m>_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMEVCNTR<m>_EL0 to 0. On reads, returns the unsigned overflow flag for PMEVCNTR<m>_EL0 overflow status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 has not overflowed.
0b1	PMEVCNTR<m>_EL0 has overflowed.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>_EL0[31:0] or unsigned overflow of PMEVCNTR<m>_EL0[63:0].

When FEAT_PMUv3_EXTPMN is implemented, MDCR_EL2.HLP and PMCR_EL0.LP are applicable only for event counters in the second and first range. For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.

        - m >= UInt(EffectiveEPMN()).
        - !IsMostSecureAccess(addrdesc).
    - When SoftwareLockStatus(), access to this field is **RO**.
    - Otherwise, access to this field is **W1C**.

Accessing PMOVSCLR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented

[63:0] Accessible at offset `0xC80` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXT32 is implemented and SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented

[31:0] Accessible at offset `0xC80` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVSSET_EL0, Performance Monitors Overflow Flag Status Set Register

The PMOVSSET_EL0 characteristics are:

Purpose

Allows software to set the unsigned overflow flags for the following counters to 1:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows the current unsigned overflow flag values.

Configuration

External register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMOVSSET_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented.

External register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMOVSCLR_EL0[31:0] when FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented.

External register PMOVSSET_EL0 bits [63:32] are architecturally mapped to AArch64 System register PMOVSSET_EL0[63:32] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMOVSSET_EL0 bits [63:32] are architecturally mapped to AArch64 System register PMOVSCLR_EL0[63:32] when FEAT_PMUv3_EXT64 is implemented or FEAT_PMUv3p9 is implemented.

External register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSR[31:0].

External register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSSET[31:0].

This register is present only when FEAT_PMUv3_EXT is implemented. Otherwise, direct accesses to PMOVSSET_EL0 are RES0.

PMOVSSET_EL0 is in the Core power domain.

Attributes

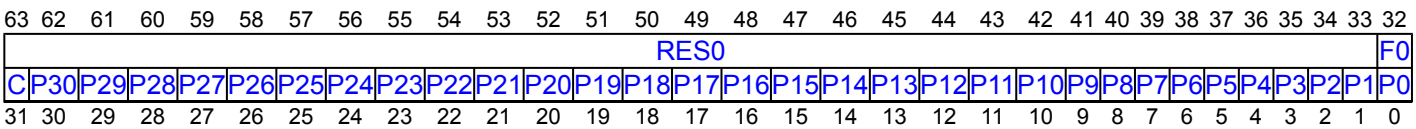
PMOVSSET_EL0 is a:

- 64-bit register when FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented
- 32-bit register otherwise

This register is part of the PMU block.

Field descriptions

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3p9 is implemented, or FEAT_PMUv3_ICNTR is implemented:



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Unsigned overflow flag for PMICNTR_EL0 set. On writes, allows software to set the unsigned overflow flag for PMICNTR_EL0 to 1. On reads, returns the unsigned overflow flag for PMICNTR_EL0.

F0	Meaning
0b0	PMICNTR_EL0 has not overflowed.
0b1	PMICNTR_EL0 has overflowed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

Otherwise:

Reserved, RES0.

C, bit [31]

Unsigned overflow flag for PMCCNTR_EL0 set. On writes, allows software to set the unsigned overflow flag for PMCCNTR_EL0 to 1. On reads, returns the unsigned overflow flag for PMCCNTR_EL0 overflow status.

C	Meaning
0b0	PMCCNTR_EL0 has not overflowed.
0b1	PMCCNTR_EL0 has overflowed.

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

Unsigned overflow flag for PMEVCNTR<m>_EL0 set. On writes, allows software to set the unsigned overflow flag for PMEVCNTR<m>_EL0 to 1. On reads, returns the unsigned overflow flag for PMEVCNTR<m>_EL0 overflow status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 has not overflowed.
0b1	PMEVCNTR<m>_EL0 has overflowed.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>_EL0[31:0] or unsigned overflow of PMEVCNTR<m>_EL0[63:0].

When FEAT_PMUv3_EXTPMN is implemented, MDCR_EL2.HLP and PMCR_EL0.LP are applicable only for event counters in the second and first range. For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).
  - !IsMostSecureAccess(addrdesc).
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

Otherwise:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

C, bit [31]

Unsigned overflow flag for PMCCNTR_EL0 set. On writes, allows software to set the unsigned overflow flag for PMCCNTR_EL0 to 1. On reads, returns the unsigned overflow flag for PMCCNTR_EL0 overflow status.

C	Meaning
0b0	PMCCNTR_EL0 has not overflowed.
0b1	PMCCNTR_EL0 has overflowed.

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **W1S**.

P<m>, bit [m], for m = 30 to 0

Unsigned overflow flag for PMEVCNTR<m>_EL0 set. On writes, allows software to set the unsigned overflow flag for PMEVCNTR<m>_EL0 to 1. On reads, returns the unsigned overflow flag for PMEVCNTR<m>_EL0 overflow status.

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 has not overflowed.
0b1	PMEVCNTR<m>_EL0 has overflowed.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>_EL0[31:0] or unsigned overflow of PMEVCNTR<m>_EL0[63:0].

When FEAT_PMUv3_EXTPMN is implemented, MDCR_EL2.HLP and PMCR_EL0.LP are applicable only for event counters in the second and first range. For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Cold reset, when FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.

        - m >= UInt(EffectiveEPMN()).
        - !IsMostSecureAccess(addrdesc).
    - When SoftwareLockStatus(), access to this field is **RO**.
    - Otherwise, access to this field is **W1S**.

Accessing PMOVSSET_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_ICNTR is implemented, or FEAT_PMUv3p9 is implemented

[63:0] Accessible at offset `0xCC0` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When FEAT_PMUv3_EXT32 is implemented and SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

When FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3_ICNTR is not implemented, and FEAT_PMUv3p9 is not implemented

[31:0] Accessible at offset `0xCC0` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **RO**.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPCSCTL, PC Sample-based Profiling Control Register

The PMPCSCTL characteristics are:

Purpose

Controls the PC Sample-based Profiling feature.

Configuration

This register is present only when FEAT_PCSRv8p9 is implemented. Otherwise, direct accesses to PMPCSCTL are RES0.

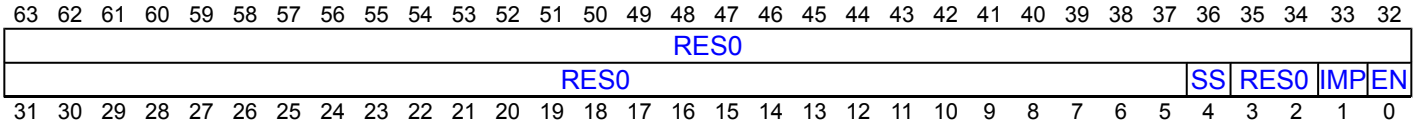
PMPCSCTL is in the Core power domain.

Attributes

PMPCSCTL is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:5]

Reserved, RES0.

SS, bit [4]

When FEAT_PMUv3_SS is implemented:

Sample on Snapshot.

Controls whether the following registers are sampled on a PMU snapshot Capture event:

- If FEAT_PMUv3_EXT32 is implemented: PMCID1SR, PMCID2SR, PMPCSR, and PMVIDSR.
- If FEAT_PMUv3_EXT64 is implemented: PMCCIDSR, PMPCSR, and PMVCIDSR.

SS	Meaning
0b0	Sample on Read.
0b1	Sample on Snapshot.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bits [3:2]

Reserved, RES0.

IMP, bit [1]

Profiling enable implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IMP	Meaning
0b0	PMPCSCTL.EN reads-as-zero and ignores writes.
0b1	PMPCSCTL.EN is a read-write control bit.

Access to this field is **RO**.

EN, bit [0]

When PMU.PMPCSCTL.IMP == 1:

PC Sample-based Profiling Enable.

EN	Meaning
0b0	PC Sample-based Profiling is suspended.
0b1	PC Sample-based Profiling is active.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RAZ/WI.

Accessing PMPCSCTL

Accesses to this register use the following encodings:

Accessible at offset 0xE50 from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPCSR, Program Counter Sample Register

The PMPCSR characteristics are:

Purpose

Holds a sampled instruction address value.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and FEAT_PCSRv8p2 is implemented. Otherwise, direct accesses to PMPCSR are RES0.

PMPCSR is in the Core power domain.

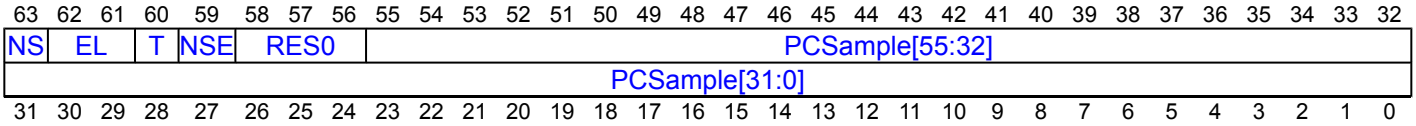
If FEAT_PMUv3_EXT32 is implemented, support for 64-bit atomic reads is IMPLEMENTATION DEFINED. If 64-bit atomic reads are implemented, a 64-bit read of PMPCSR has the same side-effect as a 32-bit read of PMCSR[31:0] followed by a 32-bit read of PMPCSR[63:32], returning the combined value. For example, if the PE is in Debug state then a 64-bit atomic read returns bits[31:0] == `0xFFFFFFFF` and bits[63:32] UNKNOWN.

Attributes

PMPCSR is a 64-bit register.

This register is part of the PMU block.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

Together with the NSE field, indicates the Security state that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

NSE	NS	Meaning
`0b0`	`0b0`	When Secure state is implemented, Secure. Otherwise reserved.
`0b0`	`0b1`	Non-secure.
`0b1`	`0b0`	Root.
`0b1`	`0b1`	Realm.

Otherwise:

Non-secure state sample. Indicates the Security state that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

If EL3 is not implemented, this bit indicates the Effective value of SCR.NS.

NS	Meaning
`0b0`	Sample is from Secure state.
`0b1`	Sample is from Non-secure state.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

EL, bits [62:61]

Exception level status sample. Indicates the Exception level that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

EL	Meaning
0b00	Sample is from EL0.
0b01	Sample is from EL1.
0b10	Sample is from EL2.
0b11	Sample is from EL3.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

T, bit [60]

When FEAT_TME is implemented:

Transactional state of the sample. Indicates the Transactional state that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

T	Meaning
0b0	Sample is from Non-transactional state.
0b1	Sample is from Transactional state.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSE, bit [59]

When FEAT_RME is implemented:

Together with the NS field, indicates the Security state that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

For a description of the values derived by evaluating NS and NSE together, see PMPCSR.NS.

Otherwise:

Reserved, RES0.

Bits [58:56]

Reserved, RES0.

PCSample[55:32], bits [55:32]

Bits[55:32] of the sampled instruction address value. The translation regime that PMPCSR samples can be determined from PMPCSR.{NS,EL}.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PCSample[31:0], bits [31:0]

Bits[31:0] of the sampled instruction address value.

PMPCSR[31:0] reads as `0xFFFFFFFF` when any of the following are true:

- The PE is in Debug state.
- PC Sample-based profiling is prohibited.

If a branch instruction has retired since the PE left reset state, then the first read of PMPCSR[31:0] is permitted but not required to return `0xFFFFFFFF`.

PMPCSR[31:0] reads as an UNKNOWN value when any of the following are true:

- The PE is in reset state.
- No branch instruction has retired since the PE left reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No branch instruction has retired since the last read of PMPCSR[31:0].

For the cases where a read of PMPCSR[31:0] returns `0xFFFFFFFF` or an UNKNOWN value, the read has the side-effect of setting PMPCSR[63:32], PMCID1SR, PMCID2SR, and PMVIDSR to UNKNOWN values.

Otherwise, a read of PMPCSR[31:0] returns bits [31:0] of the sampled instruction address value and has the side-effect of indirectly writing to PMPCSR[63:32], PMCID1SR, PMCID2SR, and PMVIDSR. The translation regime that PMPCSR samples can be determined from PMPCSR.{NS,EL}.

For a read of PMPCSR[31:0] from the memory-mapped interface, if PMLSR.SLK == 1, meaning the OPTIONAL Software Lock is locked, then the side-effect of the access does not occur and PMPCSR[63:32], PMCID1SR, PMCID2SR, and PMVIDSR are unchanged.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing PMPCSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

Note

A 32-bit access to PMPCSR[63:32] does not update the PC sample registers. Only a 64-bit access to PMPCSR[63:0] or a 32-bit access to PMPCSR[31:0] updates the PC sample registers. This includes the value a subsequent 32-bit read of PMPCSR[63:32] will return.

Accesses to this register use the following encodings:

When FEAT_PMUv3_EXT64 is implemented

[63:0] Accessible at offset `0x200` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

When FEAT_PMUv3_EXT32 is implemented

[31:0] Accessible at offset `0x200` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

When FEAT_PMUv3_EXT32 is implemented

[63:32] Accessible at offset `0x204` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

When FEAT_PMUv3_EXT64 is implemented

[63:0] Accessible at offset `0x220` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

When FEAT_PMUv3_EXT32 is implemented

[31:0] Accessible at offset `0x220` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

When FEAT_PMUv3_EXT32 is implemented

[63:32] Accessible at offset `0x224` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPIDR0, Performance Monitors Peripheral Identification Register 0

The PMPIDR0 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMPIDR0. Otherwise, direct accesses to PMPIDR0 are RES0.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

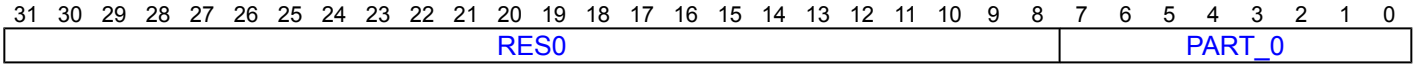
This register is required for CoreSight compliance.

Attributes

PMPIDR0 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, least significant byte.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMPIDR0

Accesses to this register use the following encodings:

Accessible at offset `0xFE0` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPIDR1, Performance Monitors Peripheral Identification Register 1

The PMPIDR1 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMPIDR1. Otherwise, direct accesses to PMPIDR1 are RES0.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

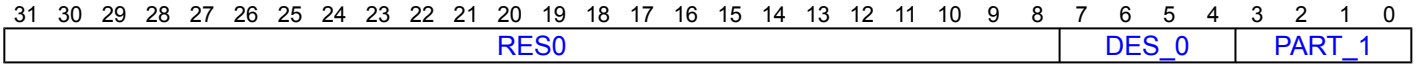
This register is required for CoreSight compliance.

Attributes

PMPIDR1 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, least significant nibble of JEP106 ID code. For Arm Limited, this field is `0b1011`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, most significant nibble.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMPIDR1

Accesses to this register use the following encodings:

Accessible at offset `0xFE4` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPIDR2, Performance Monitors Peripheral Identification Register 2

The PMPIDR2 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMPIDR2. Otherwise, direct accesses to PMPIDR2 are RES0.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

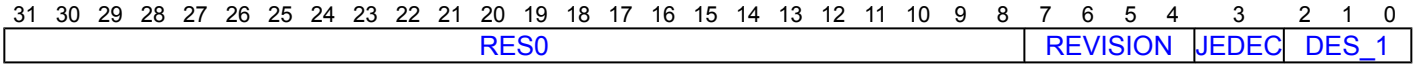
This register is required for CoreSight compliance.

Attributes

PMPIDR2 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Part major revision. Parts can also use this field to extend Part number to 16-bits.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

Indicates a JEP106 identity code is used.

Reads as `0b1`.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, most significant bits of JEP106 ID code. For Arm Limited, this field is `0b011`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMPIDR2

Accesses to this register use the following encodings:

Accessible at offset `0xFE8` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPIDR3, Performance Monitors Peripheral Identification Register 3

The PMPIDR3 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMPIDR3. Otherwise, direct accesses to PMPIDR3 are RES0.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

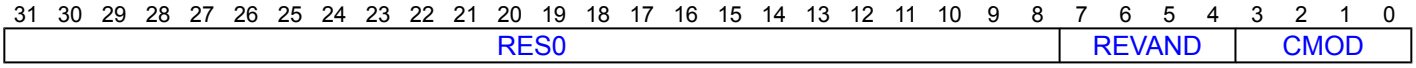
This register is required for CoreSight compliance.

Attributes

PMPIDR3 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Part minor revision. Parts using PMPIDR2.REVISION as an extension to the Part number must use this field as a major revision number.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer modified. Indicates someone other than the Designer has modified the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMPIDR3

Accesses to this register use the following encodings:

Accessible at offset `0xFEC` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPIDR4, Performance Monitors Peripheral Identification Register 4

The PMPIDR4 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

This register is present only when FEAT_PMUv3_EXT is implemented and an implementation implements PMPIDR4. Otherwise, direct accesses to PMPIDR4 are RES0.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

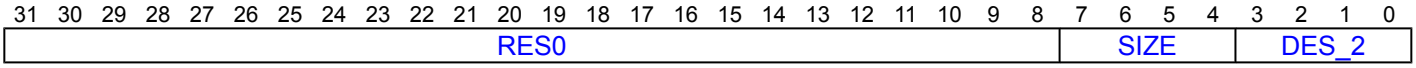
This register is required for CoreSight compliance.

Attributes

PMPIDR4 is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component. Log_2 of the number of 4KB pages from the start of the component to the end of the component ID registers.

Reads as `0b0000`.

Access to this field is **RO**.

DES_2, bits [3:0]

Designer, JEP106 continuation code, least significant nibble. For Arm Limited, this field is `0b0100`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMPIDR4

Accesses to this register use the following encodings:

Accessible at offset `0xFD0` from PMU

- When FEAT_DoPD is implemented and !IsCorePowered(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSSCR_EL1, Performance Monitors Snapshot Status and Capture Register

The PMSSCR_EL1 characteristics are:

Purpose

Holds status information about the captured counters and provides a mechanism for software to initiate a sample.

Configuration

External register PMSSCR_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMSSCR_EL1[63:0].

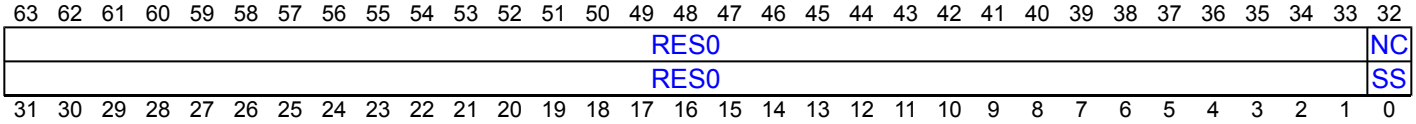
This register is present only when FEAT_PMUv3_SS is implemented. Otherwise, direct accesses to PMSSCR_EL1 are RES0.

Attributes

PMSSCR_EL1 is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:33]

Reserved, RES0.

NC, bit [32]

No Capture. Indicates whether the PMU counters have been captured.

NC	Meaning
0b0	PMU counters captured.
0b1	PMU counters not captured.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

Bits [31:1]

Reserved, RES0.

SS, bit [0]

Snapshot Capture and Status.

SS	Meaning
0b0	On a read, the Capture event has completed.
0b1	On a read, the Capture event has not completed.
	On a write, request a Capture event.

A write of 0 to this field is ignored.

It is CONSTRAINED UNPREDICTABLE whether a Capture event has completed if this field is modified when the Capture event is ongoing.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- When PMU capture events are disabled, access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Accessing PMSSCR_EL1

Accesses to this register use the following encodings:

Accessible at offset 0xE30 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMSSAccess(addrdesc), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RW**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSWINC_EL0, Performance Monitors Software Increment Register

The PMSWINC_EL0 characteristics are:

Purpose

Increments a counter that is configured to count the Software increment event, event `0x00`. For more information, see 'SW_INCR'.

Configuration

External register PMSWINC_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMSWINC_EL0[31:0].

External register PMSWINC_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMSWINC[31:0].

This register is present only when FEAT_PMUv3_EXT32 is implemented, FEAT_PMUv3p9 is not implemented, and an implementation implements PMSWINC_EL0.

PMSWINC_EL0 is in the Core power domain.

If this register is implemented, use of it is deprecated.

If 1 is written to bit [n] from the external debug interface, it is CONSTRAINED UNPREDICTABLE whether or not a SW_INCR event is created for counter n. This is consistent with not implementing the register in the external debug interface.

Attributes

PMSWINC_EL0 is a 32-bit register.

This register is part of the PMU block.

Field descriptions

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES0	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

Bit [31]

Reserved, RES0.

P<m>, bit [m], for m = 30 to 0

Event counter software increment bit for PMEVCNTR<m>_EL0.

P<m>	Meaning
`0b0`	No action. The write to this bit is ignored.
`0b1`	It is CONSTRAINED UNPREDICTABLE whether a SW_INCR event is generated for event counter m.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- When SoftwareLockStatus(), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **WO/RAZ**.

Accessing PMSWINC_EL0

If FEAT_PMUv3p9 or FEAT_PMUv3_EXT64 are implemented, this location is used for accesses to PMZR_EL0.

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

Accesses to this register use the following encodings:

Accessible at offset `0xCA0` from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **WI**.
- Otherwise, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMVCIDSR, CONTEXTIDR_EL1 and VMID Sample Register

The PMVCIDSR characteristics are:

Purpose

Contains the sampled CONTEXTIDR_EL1 and VMID values that are captured on reading PMPCSR.

Configuration

External register PMVCIDSR bits [31:0] are architecturally mapped to External register PMCCIDSR[31:0].

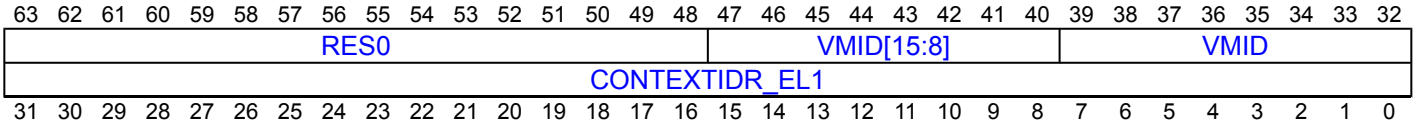
This register is present only when FEAT_PMUv3_EXT64 is implemented and FEAT_PCSRv8p2 is implemented.

Attributes

PMVCIDSR is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VMID16 is implemented:

Extension to VMID[7:0]. For more information, see VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID, bits [39:32]

VMID sample. The VMID associated with the most recent PMPCSR sample. When the most recent PMPCSR sample was generated:

- This field is set to an UNKNOWN value if any of the following apply:
  - EL2 is disabled in the current Security state.
  - The PE is executing at EL2.
  - The PE is executing at EL0, and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.
- Otherwise:
  - If EL2 is using AArch64 and either FEAT_VMID16 is not implemented or VTCR_EL2.VS is 1, this field is set to VTTBR_EL2.VMID.

    - If EL2 is using AArch64, FEAT_VMID16 is implemented, and VTCR_EL2.VS is 0, PMVIDSR.VMID[7:0] is set to VTTBR_EL2.VMID[7:0] and PMVIDSR.VMID[15:8] is RES0.
    - If EL2 is using AArch32, this field is set to VTTBR.VMID.

Because the value written to PMVIDSR is an indirect read of the VMID value, it is CONSTRAINED UNPREDICTABLE whether PMVIDSR is set to the original or new value if PMPCSR samples:

- An instruction that writes to the VMID value.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CONTEXTIDR_EL1, bits [31:0]

Context ID. The value of CONTEXTIDR that is associated with the most recent PMPCSR sample. When the most recent PMPCSR sample is generated:

- If EL1 is using AArch64, then the Context ID is sampled from CONTEXTIDR_EL1.
- If EL1 is using AArch32, then the Context ID is sampled from CONTEXTIDR.
- If EL3 is implemented and is using AArch32, then CONTEXTIDR is a banked register and this register samples the current banked copy of CONTEXTIDR for the Security state that is associated with the most recent PMPCSR sample.

Because the value written to this register is an indirect read of CONTEXTIDR, it is CONSTRAINED UNPREDICTABLE whether this register is set to the original or new value if PMPCSR samples:

- An instruction that writes to CONTEXTIDR.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing PMVCIDSR

If FEAT_PCSRv8p2 and FEAT_PMUv3_EXT32 are implemented, then the same content is present in the same locations, and can be accessed using PMVIDSR[31:0] and PMCID1SR[31:0].

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

Accesses to this register use the following encodings:

Accessible at offset `0x208` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMVIDSR, VMID Sample Register

The PMVIDSR characteristics are:

Purpose

Contains the sampled VMID value that is captured on reading PMPCSR[31:0].

Configuration

This register is present only when FEAT_PMUv3_EXT32 is implemented, FEAT_PCSRv8p2 is implemented, and EL2 is implemented.

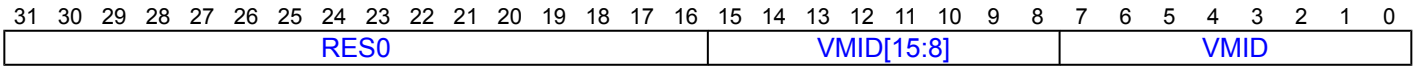
PMVIDSR is in the Core power domain.

Attributes

PMVIDSR is a 32-bit register.

This register is part of the PMU block.

Field descriptions



Bits [31:16]

Reserved, RES0.

VMID[15:8], bits [15:8]

When FEAT_VMID16 is implemented:

Extension to VMID[7:0]. For more information, see VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID, bits [7:0]

VMID sample. The VMID associated with the most recent PMPCSR sample. When the most recent PMPCSR sample was generated:

- This field is set to an UNKNOWN value if any of the following apply:
  - EL2 is disabled in the current Security state.
  - The PE is executing at EL2.
  - The PE is executing at EL0, and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.
- Otherwise:
  - If EL2 is using AArch64 and either FEAT_VMID16 is not implemented or VTCR_EL2.VS is 1, this field is set to VTTBR_EL2.VMID.
  - If EL2 is using AArch64, FEAT_VMID16 is implemented, and VTCR_EL2.VS is 0, PMVIDSR.VMID[7:0] is set to VTTBR_EL2.VMID[7:0] and PMVIDSR.VMID[15:8] is RES0.

- If EL2 is using AArch32, this field is set to VTTBR.VMID.

Because the value written to PMVIDSR is an indirect read of the VMID value, it is CONSTRAINED UNPREDICTABLE whether PMVIDSR is set to the original or new value if PMPCSR samples:

- An instruction that writes to the VMID value.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing PMVIDSR

If FEAT_PMUv3_EXT64 is implemented, then the same content is present in the same location, and can be accessed using PMVCIDSR[63:32].

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

Accesses to this register use the following encodings:

Accessible at offset `0x20C` from PMU

- When DoubleLockStatus() or !IsCorePowered(), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMZR_EL0, Performance Monitors Zero with Mask

The PMZR_EL0 characteristics are:

Purpose

Zero the set of counters specified by the mask written to PMZR_EL0.

Configuration

External register PMZR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMZR_EL0[63:0].

This register is present only when FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p9 is implemented.

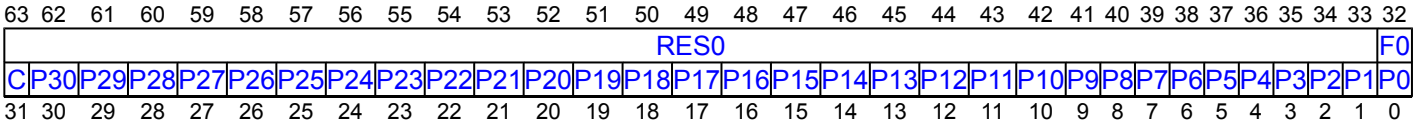
PMZR_EL0 is in the Core power domain.

Attributes

PMZR_EL0 is a 64-bit register.

This register is part of the PMU block.

Field descriptions



Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Zero PMICNTR_EL0.

F0	Meaning
0b0	Write is ignored.
0b1	Set PMICNTR_EL0 to zero.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **WO/RAZ**.

Otherwise:

Reserved, RES0.

C, bit [31]

Zero PMCCNTR_EL0.

C	Meaning
0b0	Write is ignored.
0b1	Set PMCCNTR_EL0 to zero.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **WO/RAZ**.

P<m>, bit [m], for m = 30 to 0

Zero PMEVCNTR<m>_EL0.

P<m>	Meaning
0b0	Write is ignored.
0b1	Set PMEVCNTR<m>_EL0 to zero.

Accessing this field has the following behavior:

- When m >= NUM_PMU_COUNTERS, access to this field is **RAZ/WI**.
- Access to this field is **RAZ/WI** if all the following are true:
  - FEAT_PMUv3_EXTPMN is implemented.
  - m >= UInt(EffectiveEPMN()).
  - !IsMostSecureAccess(addrdesc).
- When SoftwareLockStatus(), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **WO/RAZ**.

Accessing PMZR_EL0

When FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p9 is not implemented, then this location might be used for PMSWINC_EL0. If PMSWINC_EL0 is not implemented, then accesses to this location are RES0.

Accesses to this register use the following encodings:

Accessible at offset 0xCA0 from PMU

- When DoubleLockStatus(), or !IsCorePowered(), or !AllowExternalPMUAccess(addrdesc), accesses to this register generate an error response.
- When (FEAT_PMUv3_EXTPMN is not implemented, or !IsMostSecureAccess(addrdesc), or PMCCR.OSLO == 0) and OSLockStatus(), accesses to this register generate an error response.
- When SoftwareLockStatus(), accesses to this register are **WI**.
- Otherwise, accesses to this register are **WO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMU

The AMU characteristics are:

Attributes

AMU is a block of size: 4096 bytes

Contents

Offset	Name	Accessor condition	Register condition	Most permissive access
0x000 + (8 * n) for n in 16:0	AMEVCNTR0<n>	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented	RO
0x000 + (8 * n) for n in 16:0	AMEVCNTR0<n>	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented	RO
0x100 + (8 * n) for n in 16:0	AMEVCNTR1<n>	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented	RO
0x100 + (8 * n) for n in 16:0	AMEVCNTR1<n>	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented	RO
0x400 + (8 * n) for n in 16:0	AMEVTYPER0<n>	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented	RO
0x400 + (4 * n) for n in 16:0	AMEVTYPER0<n>	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented	RO
0x480 + (4 * n) for n in 16:0	AMEVTYPER1<n>	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented	RO
0x500 + (8 * n) for n in 16:0	AMEVTYPER1<n>	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented	RO
0xC00	AMCNTENSET	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented and FEAT_AMU_EXT64 is implemented	RO
0xC00	AMCNTENSET0	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented and FEAT_AMU_EXT32 is implemented	RO
0xC04	AMCNTENSET1	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented and FEAT_AMU_EXT32 is implemented	RO
0xC10	AMCNTEN	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented and FEAT_AMU_EXT64 is implemented	RO

0xC20	AMCNTENCLR	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented and FEAT_AMU_EXT64 is implemented	RO
0xC20	AMCNTENCLR0	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented and FEAT_AMU_EXT32 is implemented	RO
0xC24	AMCNTENCLR1	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented	RO
0xCE0	AMCGCR	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented	RO
0xCE0	AMCGCR	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented	RO
0xE00	AMCFGR	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented	RO
0xE00	AMCFGR	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented	RO
0xE04	AMCR	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented	RO
0xE08	AMIIDR	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented	RO
0xE08	AMIIDR	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented	RO
0xE10	AMCR	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented	RO
0xE40	AMSCR	When FEAT_AMU_EXTACR is implemented and FEAT_RME is not implemented	When FEAT_AMU_EXTACR is implemented and FEAT_RME is not implemented	RW
0xE48	AMROOTCR	When FEAT_AMU_EXTACR is implemented and FEAT_RME is implemented	When FEAT_AMU_EXTACR is implemented and FEAT_RME is implemented	RW
0xFA8	AMDEVAFF	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented, FEAT_AMU_EXT64 is implemented, and an implementation implements AMDEVAFF1	RO
0xFA8	AMDEVAFF0	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented, FEAT_AMU_EXT32 is implemented, and an implementation implements AMDEVAFF0	RO

0xFAC	AMDEVAFF1	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented, FEAT_AMU_EXT32 is implemented, and an implementation implements AMDEVAFF1	RO
0xFBC	AMDEVARCH	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMDEVARCH	RO
0xFBC	AMDEVARCH	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMDEVARCH	RO
0xFCC	AMDEVTYPE	When FEAT_AMU_EXT64 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMDEVTYPE	RO
0xFCC	AMDEVTYPE	When FEAT_AMU_EXT32 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMDEVTYPE	RO
0xFD0	AMPIDR4	When FEAT_AMUv1 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMPIDR4	RO
0xFE0	AMPIDR0	When FEAT_AMUv1 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMPIDR0	RO
0xFE4	AMPIDR1	When FEAT_AMUv1 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMPIDR1	RO
0xFE8	AMPIDR2	When FEAT_AMUv1 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMPIDR2	RO
0xFEC	AMPIDR3	When FEAT_AMUv1 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMPIDR3	RO
0xFF0	AMCIDR0	When FEAT_AMUv1 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMCIDR0	RO
0xFF4	AMCIDR1	When FEAT_AMUv1 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMCIDR1	RO
0xFF8	AMCIDR2	When FEAT_AMUv1 is implemented	When FEAT_AMUv1 is implemented and an implementation implements AMCIDR2	RO
0xFFC	AMCIDR3	When FEAT_AMUv1 is implemented	When FEAT_AMUv1 is implemented and an	RO

			implementation implements AMCIDR3	

Direct accesses to other offsets in this block are RES0.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCFGR, Activity Monitors Configuration Register

The AMCFGR characteristics are:

Purpose

Global configuration register for the activity monitors.

Provides information on supported features, the number of counter groups implemented, the total number of activity monitor event counters implemented, and the size of the counters. AMCFGR is applicable to both the architected and the auxiliary counter groups.

Configuration

External register AMCFGR bits [31:0] are architecturally mapped to AArch64 System register AMCFGR_EL0[31:0] when FEAT_AMU_EXT32 is implemented.

External register AMCFGR bits [63:0] are architecturally mapped to AArch64 System register AMCFGR_EL0[63:0] when FEAT_AMU_EXT64 is implemented.

External register AMCFGR bits [31:0] are architecturally mapped to AArch32 System register AMCFGR[31:0].

It is IMPLEMENTATION DEFINED whether AMCFGR is implemented in the Core power domain or in the Debug power domain.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCFGR are RES0.

Attributes

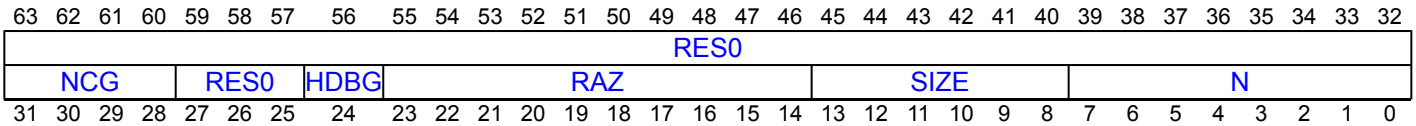
AMCFGR is a:

- 64-bit register when FEAT_AMU_EXT64 is implemented
- 32-bit register otherwise

This register is part of the AMU block.

Field descriptions

When FEAT_AMU_EXT64 is implemented:



Bits [63:32]

Reserved, RES0.

NCG, bits [31:28]

Defines the number of counter groups implemented, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCG	Meaning
0b0000	One counter group implemented.
0b0001	Two counter groups implemented.

All other values are reserved.

Access to this field is **RO**.

Bits [27:25]

Reserved, RES0.

HDBG, bit [24]

Halt-on-debug supported.

This feature must be supported, and so this bit is `0b1`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HDBG	Meaning
`0b0`	AMCR.HDBG is RES0.
`0b1`	AMCR.HDBG is read/write.

Access to this field is **RO**.

Bits [23:14]

Reserved, RAZ.

SIZE, bits [13:8]

Defines the size of the activity monitor event counters, minus one.

The counters are 64-bit, so the value of this field is `0b111111`.

This field is used by software to determine the spacing of the counters in the memory-map. The counters are at doubleword-aligned addresses.

Reads as `0b111111`.

Access to this field is **RO**.

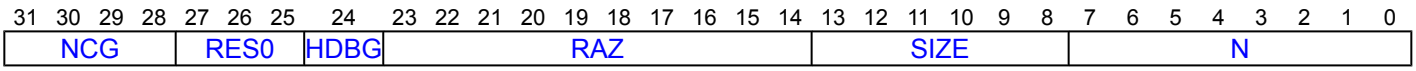
N, bits [7:0]

Defines the number of activity monitor event counters implemented in all groups, minus one.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:



NCG, bits [31:28]

Defines the number of counter groups implemented, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCG	Meaning
`0b0000`	One counter group implemented.
`0b0001`	Two counter groups implemented.

All other values are reserved.

Access to this field is **RO**.

Bits [27:25]

Reserved, RES0.

HDBG, bit [24]

Halt-on-debug supported.

This feature must be supported, and so this bit is `0b1`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HDBG	Meaning
`0b0`	AMCR.HDBG is RES0.
`0b1`	AMCR.HDBG is read/write.

Access to this field is **RO**.

Bits [23:14]

Reserved, RAZ.

SIZE, bits [13:8]

Defines the size of the activity monitor event counters, minus one.

The counters are 64-bit, so the value of this field is `0b111111`.

This field is used by software to determine the spacing of the counters in the memory-map. The counters are at doubleword-aligned addresses.

Reads as `0b111111`.

Access to this field is **RO**.

N, bits [7:0]

Defines the number of activity monitor event counters implemented in all groups, minus one.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMCFGR

Accesses to this register use the following encodings:

When FEAT_AMU_EXT64 is implemented

Accessible at offset `0xE00` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

When FEAT_AMU_EXT32 is implemented

Accessible at offset `0xE00` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCGCR, Activity Monitors Counter Group Configuration Register

The AMCGCR characteristics are:

Purpose

Provides information on the number of activity monitor event counters implemented within each counter group.

Configuration

External register AMCGCR bits [31:0] are architecturally mapped to AArch64 System register AMCGCR_EL0[31:0] when FEAT_AMU_EXT32 is implemented.

External register AMCGCR bits [63:0] are architecturally mapped to AArch64 System register AMCGCR_EL0[63:0] when FEAT_AMU_EXT64 is implemented.

External register AMCGCR bits [31:0] are architecturally mapped to AArch32 System register AMCGCR[31:0].

It is IMPLEMENTATION DEFINED whether AMCGCR is implemented in the Core power domain or in the Debug power domain.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCGCR are RES0.

Attributes

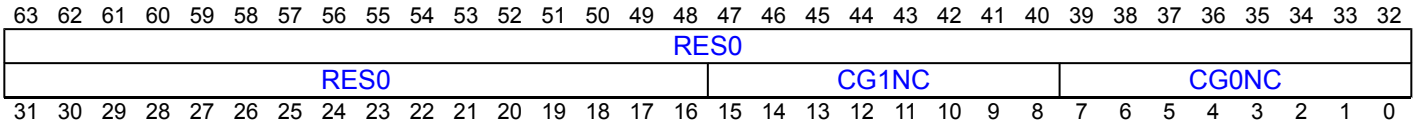
AMCGCR is a:

- 64-bit register when FEAT_AMU_EXT64 is implemented
- 32-bit register otherwise

This register is part of the AMU block.

Field descriptions

When FEAT_AMU_EXT64 is implemented:



Bits [63:16]

Reserved, RES0.

CG1NC, bits [15:8]

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

In an implementation that includes FEAT_AMUv1, the permitted range of values is 0 to 16.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

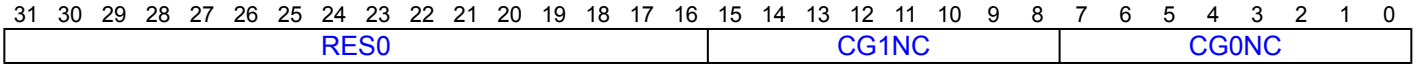
CG0NC, bits [7:0]

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

Reads as `0x04`.

Access to this field is **RO**.

Otherwise:



Bits [31:16]

Reserved, RES0.

CG1NC, bits [15:8]

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

In an implementation that includes FEAT_AMUv1, the permitted range of values is 0 to 16.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CG0NC, bits [7:0]

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

Reads as `0x04`.

Access to this field is **RO**.

Accessing AMCGCR

Accesses to this register use the following encodings:

When FEAT_AMU_EXT64 is implemented

Accessible at offset `0xCE0` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

When FEAT_AMU_EXT32 is implemented

Accessible at offset `0xCE0` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCIDR0, Activity Monitors Component Identification Register 0

The AMCIDR0 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Component identification scheme'.

Configuration

It is IMPLEMENTATION DEFINED whether AMCIDR0 is implemented in the Core power domain or in the Debug power domain.

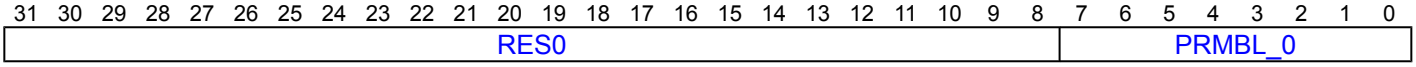
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMCIDR0. Otherwise, direct accesses to AMCIDR0 are RES0.

Attributes

AMCIDR0 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Preamble.

Reads as 0x0D.

Access to this field is **RO**.

Accessing AMCIDR0

Accesses to this register use the following encodings:

Accessible at offset 0xFF0 from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCIDR1, Activity Monitors Component Identification Register 1

The AMCIDR1 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Component identification scheme'.

Configuration

It is IMPLEMENTATION DEFINED whether AMCIDR1 is implemented in the Core power domain or in the Debug power domain.

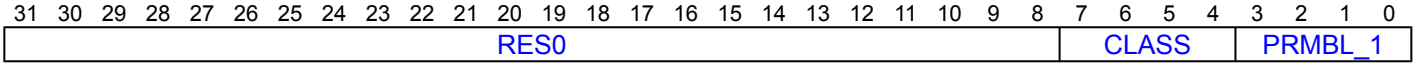
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMCIDR1. Otherwise, direct accesses to AMCIDR1 are RES0.

Attributes

AMCIDR1 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

CLASS	Meaning
0b1001	CoreSight component.

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

Access to this field is **RO**.

PRMBL_1, bits [3:0]

Preamble.

Reads as 0b0000.

Access to this field is **RO**.

Accessing AMCIDR1

Accesses to this register use the following encodings:

Accessible at offset `0xFF4` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCIDR2, Activity Monitors Component Identification Register 2

The AMCIDR2 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Component identification scheme'.

Configuration

It is IMPLEMENTATION DEFINED whether AMCIDR2 is implemented in the Core power domain or in the Debug power domain.

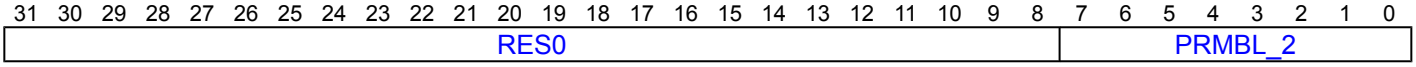
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMCIDR2. Otherwise, direct accesses to AMCIDR2 are RES0.

Attributes

AMCIDR2 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Preamble.

Reads as `0x05`.

Access to this field is **RO**.

Accessing AMCIDR2

Accesses to this register use the following encodings:

Accessible at offset `0xFF8` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCIDR3, Activity Monitors Component Identification Register 3

The AMCIDR3 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Component identification scheme'.

Configuration

It is IMPLEMENTATION DEFINED whether AMCIDR3 is implemented in the Core power domain or in the Debug power domain.

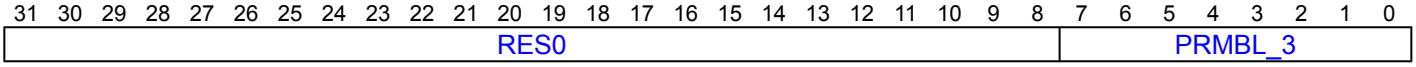
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMCIDR3. Otherwise, direct accesses to AMCIDR3 are RES0.

Attributes

AMCIDR3 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Preamble.

Reads as `0xB1`.

Access to this field is **RO**.

Accessing AMCIDR3

Accesses to this register use the following encodings:

Accessible at offset `0xFFC` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTEN, Activity Monitors Count Set and Clear Register

The AMCNTEN characteristics are:

Purpose

Control bits for the architected and auxiliary activity monitors event counters, AMEVCNTR0<n>. and AMEVCNTR1<n>.

Configuration

It is IMPLEMENTATION DEFINED whether AMCNTEN is implemented in the Core power domain or in the Debug power domain.

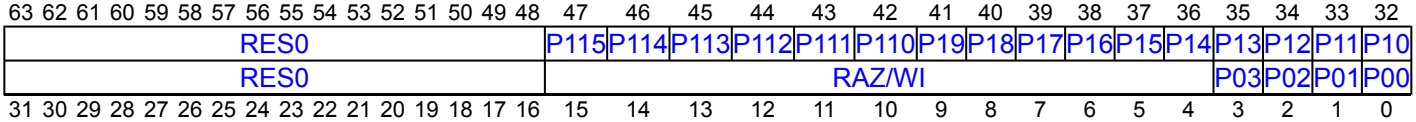
This register is present only when FEAT_AMUv1 is implemented and FEAT_AMU_EXT64 is implemented. Otherwise, direct accesses to AMCNTEN are RES0.

Attributes

AMCNTEN is a 64-bit register.

This register is part of the AMU block.

Field descriptions



Bits [63:48]

Reserved, RES0.

P1<n>, bit [n+32], for n = 15 to 0

Activity monitor event counter control bit for AMEVCNTR1<n>.

Possible values of each bit are:

P1<n>	Meaning
0b0	When read, means that AMEVCNTR1<n> is disabled.
0b1	When read, means that AMEVCNTR1<n> is enabled.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= UInt(AMU.AMCGCR.CG1NC), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P0<n>, bit [n], for n = 3 to 0

Activity monitor event counter control bit for AMEVCNTR0<n>.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

P0<n>	Meaning
0b0	When read, means that AMEVCNTR0<n> is disabled.
0b1	When read, means that AMEVCNTR0<n> is enabled.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= 4, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing AMCNTEN

If there are no auxiliary monitor event counters implemented, reads of AMCNTEN[63:32] are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR.NCG == 0b0000.

Accesses to this register use the following encodings:

Accessible at offset 0xC10 from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENCLR, Activity Monitors Count Enable Clear Register

The AMCNTENCLR characteristics are:

Purpose

Disable control bits for the architected and auxiliary activity monitors event counters, AMEVCNTR0<n> and AMEVCNTR1<n>.

Configuration

External register AMCNTENCLR bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR0_EL0[31:0].

External register AMCNTENCLR bits [63:32] are architecturally mapped to AArch64 System register AMCNTENCLR1_EL0[31:0].

It is IMPLEMENTATION DEFINED whether AMCNTENCLR is implemented in the Core power domain or in the Debug power domain.

This register is present only when FEAT_AMUv1 is implemented and FEAT_AMU_EXT64 is implemented. Otherwise, direct accesses to AMCNTENCLR are RES0.

Attributes

AMCNTENCLR is a 64-bit register.

This register is part of the AMU block.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																P115	P114	P113	P112	P111	P110	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10
RES0																RAZ/WI												P03	P02	P01	P00
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits [63:48]

Reserved, RES0.

P1<n>, bit [n+32], for n = 15 to 0

Activity monitor event counter disable bit for AMEVCNTR1<n>.

Possible values of each bit are:

P1<n>	Meaning
0b0	When read, means that AMEVCNTR1<n> is disabled.
0b1	When read, means that AMEVCNTR1<n> is enabled.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= UInt(AMU.AMCGCR.CG1NC), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P0<n>, bit [n], for n = 3 to 0

Activity monitor event counter disable bit for AMEVCNTR0<n>.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

P0<n>	Meaning
0b0	When read, means that AMEVCNTR0<n> is disabled.
0b1	When read, means that AMEVCNTR0<n> is enabled.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= 4, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing AMCNTENCLR

If there are no auxiliary monitor event counters implemented, reads of AMCNTENCLR[63:32] are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR.NCG == 0b0000.

Accesses to this register use the following encodings:

Accessible at offset 0xC20 from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENCLR0, Activity Monitors Count Enable Clear Register 0

The AMCNTENCLR0 characteristics are:

Purpose

Disable control bits for the architected activity monitors event counters, AMEVCNTR0<n>.

Configuration

External register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR0_EL0[31:0].

External register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENSET0_EL0[31:0].

External register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR0[31:0].

External register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET0[31:0].

External register AMCNTENCLR0 bits [31:0] are architecturally mapped to External register AMCNTENSET0[31:0].

It is IMPLEMENTATION DEFINED whether AMCNTENCLR0 is implemented in the Core power domain or in the Debug power domain.

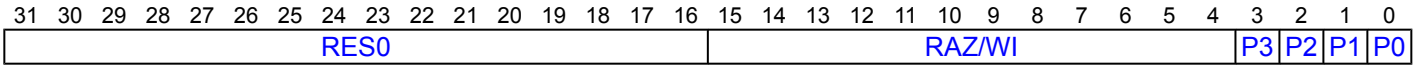
This register is present only when FEAT_AMUv1 is implemented and FEAT_AMU_EXT32 is implemented. Otherwise, direct accesses to AMCNTENCLR0 are RES0.

Attributes

AMCNTENCLR0 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter disable bit for AMEVCNTR0<n>.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR0<n> is disabled.
0b1	When read, means that AMEVCNTR0<n> is enabled.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= 4, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing AMCNTENCLR0

Accesses to this register use the following encodings:

Accessible at offset 0xC20 from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1

The AMCNTENCLR1 characteristics are:

Purpose

Disable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>.

Configuration

External register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR1_EL0[31:0].

External register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENSET1_EL0[31:0].

External register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR1[31:0].

External register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET1[31:0].

External register AMCNTENCLR1 bits [31:0] are architecturally mapped to External register AMCNTENSET1[31:0].

It is IMPLEMENTATION DEFINED whether AMCNTENCLR1 is implemented in the Core power domain or in the Debug power domain.

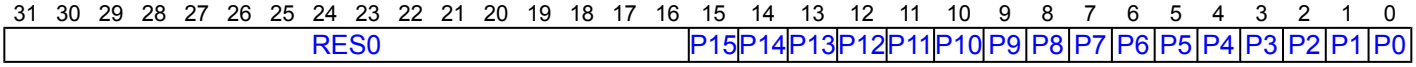
This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR1 are RES0.

Attributes

AMCNTENCLR1 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter disable bit for AMEVCNTR1<n>.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR1<n> is disabled.
0b1	When read, means that AMEVCNTR1<n> is enabled.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= UInt(AMU.AMCGCR.CG1NC), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1C**.

Accessing AMCNTENCLR1

If there are no auxiliary monitor event counters implemented, reads of AMCNTENCLR1 are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR.NCG == 0b0000.

Accesses to this register use the following encodings:

When FEAT_AMU_EXT32 is implemented

Accessible at offset 0xC24 from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET, Activity Monitors Count Enable Set Register

The AMCNTENSET characteristics are:

Purpose

Enable control bits for the architected and auxiliary activity monitors event counters, AMEVCNTR0<n> and AMEVCNTR1<n>.

Configuration

External register AMCNTENSET bits [31:0] are architecturally mapped to AArch64 System register AMCNTENSET0_EL0[31:0].

External register AMCNTENSET bits [63:32] are architecturally mapped to AArch64 System register AMCNTENSET1_EL0[31:0].

It is IMPLEMENTATION DEFINED whether AMCNTENSET is implemented in the Core power domain or in the Debug power domain.

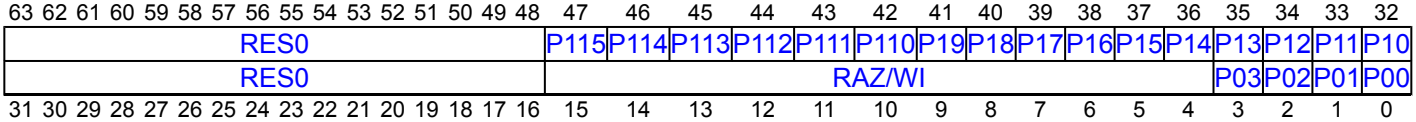
This register is present only when FEAT_AMUv1 is implemented and FEAT_AMU_EXT64 is implemented. Otherwise, direct accesses to AMCNTENSET are RES0.

Attributes

AMCNTENSET is a 64-bit register.

This register is part of the AMU block.

Field descriptions



Bits [63:48]

Reserved, RES0.

P1<n>, bit [n+32], for n = 15 to 0

Activity monitor event counter enable bit for AMEVCNTR1<n>.

Possible values of each bit are:

P1<n>	Meaning
0b0	When read, means that AMEVCNTR1<n> is disabled.
0b1	When read, means that AMEVCNTR1<n> is enabled.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= UInt(AMU.AMCGCR.CG1NC), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P0<n>, bit [n], for n = 3 to 0

Activity monitor event counter enable bit for AMEVCNTR0<n>.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

P0<n>	Meaning
0b0	When read, means that AMEVCNTR0<n> is disabled.
0b1	When read, means that AMEVCNTR0<n> is enabled.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= 4, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing AMCNTENSET

If there are no auxiliary monitor event counters implemented, reads of AMCNTENSET[63:32] are RAZ. Software must treat reserved accesses as RES0 . See 'Access requirements for reserved and unallocated registers'.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR.NCG == 0b0000.

Accesses to this register use the following encodings:

Accessible at offset 0xC00 from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET0, Activity Monitors Count Enable Set Register 0

The AMCNTENSET0 characteristics are:

Purpose

Enable control bits for the architected activity monitors event counters, AMEVCNTR0<n>.

Configuration

External register AMCNTENSET0 bits [31:0] are architecturally mapped to External register AMCNTENCLR0[31:0].

External register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR0_EL0[31:0].

External register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENSET0_EL0[31:0].

External register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR0[31:0].

External register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET0[31:0].

It is IMPLEMENTATION DEFINED whether AMCNTENSET0 is implemented in the Core power domain or in the Debug power domain.

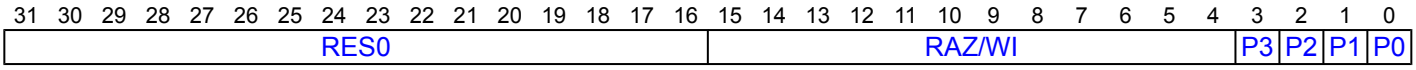
This register is present only when FEAT_AMUv1 is implemented and FEAT_AMU_EXT32 is implemented. Otherwise, direct accesses to AMCNTENSET0 are RES0.

Attributes

AMCNTENSET0 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter enable bit for AMEVCNTR0<n>.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR0<n> is disabled.
0b1	When read, means that AMEVCNTR0<n> is enabled.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= 4, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing AMCNTENSET0

Accesses to this register use the following encodings:

Accessible at offset 0xC00 from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET1, Activity Monitors Count Enable Set Register 1

The AMCNTENSET1 characteristics are:

Purpose

Enable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>.

Configuration

External register AMCNTENSET1 bits [31:0] are architecturally mapped to External register AMCNTENCLR1[31:0].

External register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR1_EL0[31:0].

External register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENSET1_EL0[31:0].

External register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR1[31:0].

External register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET1[31:0].

It is IMPLEMENTATION DEFINED whether AMCNTENSET1 is implemented in the Core power domain or in the Debug power domain.

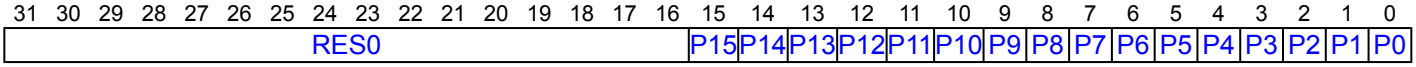
This register is present only when FEAT_AMUv1 is implemented and FEAT_AMU_EXT32 is implemented. Otherwise, direct accesses to AMCNTENSET1 are RES0.

Attributes

AMCNTENSET1 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter enable bit for AMEVCNTR1<n>.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR1<n> is disabled.
0b1	When read, means that AMEVCNTR1<n> is enabled.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= UInt(AMU.AMCGCR.CG1NC), access to this field is **RAZ/WI**.
- Otherwise, access to this field is **W1S**.

Accessing AMCNTENSET1

If there are no auxiliary monitor event counters implemented, reads of AMCNTENSET1 are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR.NCG == 0b0000.

Accesses to this register use the following encodings:

Accessible at offset 0xC04 from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCR, Activity Monitors Control Register

The AMCR characteristics are:

Purpose

Global control register for the activity monitors implementation. AMCR is applicable to both the architected and the auxiliary counter groups.

Configuration

External register AMCR bits [31:0] are architecturally mapped to AArch64 System register AMCR_EL0[31:0] when FEAT_AMU_EXT32 is implemented.

External register AMCR bits [63:0] are architecturally mapped to AArch64 System register AMCR_EL0[63:0] when FEAT_AMU_EXT64 is implemented.

External register AMCR bits [31:0] are architecturally mapped to AArch32 System register AMCR[31:0].

It is IMPLEMENTATION DEFINED whether AMCR is implemented in the Core power domain or in the Debug power domain.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCR are RES0.

Attributes

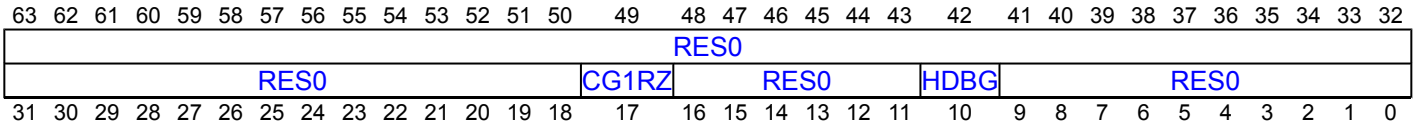
AMCR is a:

- 64-bit register when FEAT_AMU_EXT64 is implemented
- 32-bit register otherwise

This register is part of the AMU block.

Field descriptions

When FEAT_AMU_EXT64 is implemented:



Bits [63:18]

Reserved, RES0.

CG1RZ, bit [17]

When FEAT_AMUv1p1 is implemented:

Counter Group 1 Read Zero.

CG1RZ	Meaning
0b0	System register reads of AMEVCNTR1<n> return the event count at all implemented and enabled Exception levels.
0b1	If the current Exception level is the highest implemented Exception level, System register reads of AMEVCNTR1<n> return the event count. Otherwise, reads of AMEVCNTR1<n> return a zero value.

Note

Reads from the memory-mapped view of AMEVCNTR1<n> are unaffected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:11]

Reserved, RES0.

HDBG, bit [10]

This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

HDBG	Meaning
0b0	Activity monitors do not halt counting when the PE is halted in Debug state.
0b1	Activity monitors halt counting when the PE is halted in Debug state.

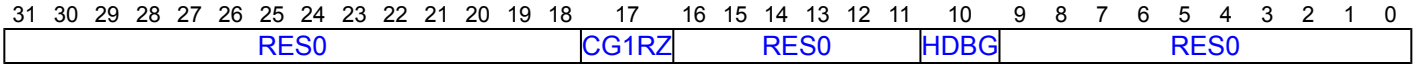
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Otherwise:



Bits [31:18]

Reserved, RES0.

CG1RZ, bit [17]

When FEAT_AMUv1p1 is implemented:

Counter Group 1 Read Zero.

CG1RZ	Meaning
0b0	System register reads of AMEVCNTR1<n> return the event count at all implemented and enabled Exception levels.
0b1	If the current Exception level is the highest implemented Exception level, System register reads of AMEVCNTR1<n> return the event count. Otherwise, reads of AMEVCNTR1<n> return a zero value.

Note

Reads from the memory-mapped view of AMEVCNTR1<n> are unaffected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:11]

Reserved, RES0.

HDBG, bit [10]

This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

HDBG	Meaning
0b0	Activity monitors do not halt counting when the PE is halted in Debug state.
0b1	Activity monitors halt counting when the PE is halted in Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing AMCR

Accesses to this register use the following encodings:

When FEAT_AMU_EXT32 is implemented

Accessible at offset 0xE04 from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

When FEAT_AMU_EXT64 is implemented

Accessible at offset 0xE10 from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMDEVAFF, Activity Monitors Device Affinity Register

The AMDEVAFF characteristics are:

Purpose

Copy of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the AMU component relates to.

Configuration

It is IMPLEMENTATION DEFINED whether AMDEVAFF is implemented in the Core power domain or in the Debug power domain.

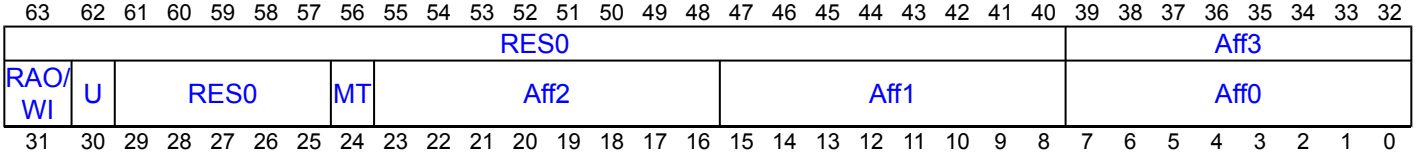
This register is present only when FEAT_AMUv1 is implemented, FEAT_AMU_EXT64 is implemented, and an implementation implements AMDEVAFF1. Otherwise, direct accesses to AMDEVAFF are RES0.

Attributes

AMDEVAFF is a 64-bit register.

This register is part of the AMU block.

Field descriptions



Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [31]

Reserved, RAO/WI.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Access to this field is **RO**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is **RO**.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMDEVAFF

Accesses to this register use the following encodings:

Accessible at offset 0xFA8 from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.

- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMDEVAFF0, Activity Monitors Device Affinity Register 0

The AMDEVAFF0 characteristics are:

Purpose

Copy of the low half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the AMU component relates to.

Configuration

It is IMPLEMENTATION DEFINED whether AMDEVAFF0 is implemented in the Core power domain or in the Debug power domain.

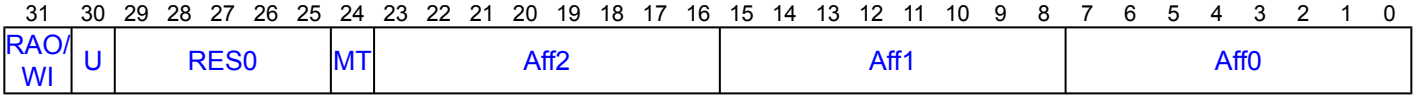
This register is present only when FEAT_AMUv1 is implemented, FEAT_AMU_EXT32 is implemented, and an implementation implements AMDEVAFF0. Otherwise, direct accesses to AMDEVAFF0 are RES0.

Attributes

AMDEVAFF0 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bit [31]

Reserved, RAO/WI.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

Access to this field is **RO**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is **RO**.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMDEVAFF0

Accesses to this register use the following encodings:

Accessible at offset 0xFA8 from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMDEVAFF1, Activity Monitors Device Affinity Register 1

The AMDEVAFF1 characteristics are:

Purpose

Copy of the high half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the AMU component relates to.

Configuration

It is IMPLEMENTATION DEFINED whether AMDEVAFF1 is implemented in the Core power domain or in the Debug power domain.

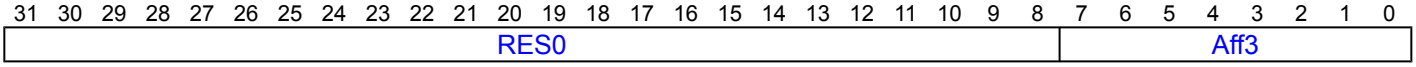
This register is present only when FEAT_AMUv1 is implemented, FEAT_AMU_EXT32 is implemented, and an implementation implements AMDEVAFF1. Otherwise, direct accesses to AMDEVAFF1 are RES0.

Attributes

AMDEVAFF1 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

Aff3, bits [7:0]

Affinity level 3. See the description of AMDEVAFF0.Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMDEVAFF1

Accesses to this register use the following encodings:

Accessible at offset `0xFAC` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMDEVARCH, Activity Monitors Device Architecture Register

The AMDEVARCH characteristics are:

Purpose

Identifies the programmers' model architecture of the AMU component.

Configuration

It is IMPLEMENTATION DEFINED whether AMDEVARCH is implemented in the Core power domain or in the Debug power domain.

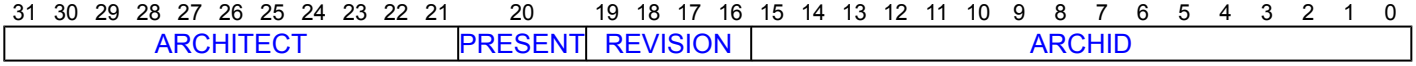
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMDEVARCH. Otherwise, direct accesses to AMDEVARCH are RES0.

Attributes

AMDEVARCH is a 32-bit register.

This register is part of the AMU block.

Field descriptions



ARCHITECT, bits [31:21]

Defines the architect of the component. For Activity Monitors, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, `0b0100`.

Bits [27:21] are the JEP106 identification code, `0b0111011`.

Reads as `0b01000111011`.

Access to this field is **RO**.

PRESENT, bit [20]

DEVARCH present. Indicates that the AMDEVARCH register is present.

Reads as `0b1`.

Access to this field is **RO**.

REVISION, bits [19:16]

Defines the architecture revision. For architectures defined by Arm this is the minor revision.

REVISION	Meaning
`0b0000`	Architecture revision is AMUv1.

All other values are reserved.

Access to this field is **RO**.

ARCHID, bits [15:0]

Defines this part to be an AMU component. For architectures defined by Arm this is further subdivided.

For AMU:

- Bits [15:12] are the architecture version, also identified as AMDEVARCH.ARCHVER.
- Bits [11:0] are the architecture part number, also identified as AMDEVARCH.ARCHPART.

AMDEVARCH.ARCHVER = `0x0`, which corresponds to AMU architecture version AMUv1.

If FEAT_AMU_EXT32 is implemented, AMDEVARCH is `0xA66`.

If FEAT_AMU_EXT64 is implemented, AMDEVARCH is `0xA67`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ARCHID	Meaning
`0x0A66`	AMUv1, with FEAT_AMU_EXT32 implemented.
`0x0A67`	AMUv1, with FEAT_AMU_EXT64 implemented.

Access to this field is **RO**.

Accessing AMDEVARCH

Accesses to this register use the following encodings:

When FEAT_AMU_EXT64 is implemented

Accessible at offset `0xFBC` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

When FEAT_AMU_EXT32 is implemented

Accessible at offset `0xFBC` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMDEVTYPE, Activity Monitors Device Type Register

The AMDEVTYPE characteristics are:

Purpose

Indicates to a debugger that this component is part of a PE's performance monitor interface.

Configuration

It is IMPLEMENTATION DEFINED whether AMDEVTYPE is implemented in the Core power domain or in the Debug power domain.

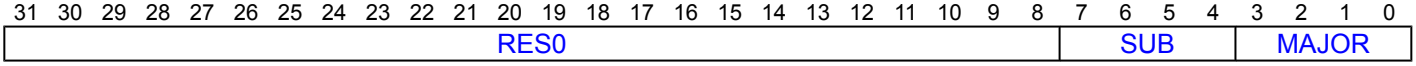
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMDEVTYPE. Otherwise, direct accesses to AMDEVTYPE are RES0.

Attributes

AMDEVTYPE is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Subtype.

SUB	Meaning
0b0001	Component within a PE.

Access to this field is **RO**.

MAJOR, bits [3:0]

Major type.

MAJOR	Meaning
0b0110	Performance monitor component.

Access to this field is **RO**.

Accessing AMDEVTYPE

Accesses to this register use the following encodings:

When FEAT_AMU_EXT64 is implemented

Accessible at offset 0xFCC from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

When FEAT_AMU_EXT32 is implemented

Accessible at offset `0xFCC` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTR0<n>, Activity Monitors Event Counter Registers 0, n = 0 - 3

The AMEVCNTR0<n> characteristics are:

Purpose

Provides access to the architected activity monitor event counters.

Configuration

External register AMEVCNTR0<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVCNTR0<n>_EL0[63:0].

External register AMEVCNTR0<n> bits [31:0] are architecturally mapped to AArch32 System register AMEVCNTR0<n>[31:0].

It is IMPLEMENTATION DEFINED whether AMEVCNTR0<n> is implemented in the Core power domain or in the Debug power domain.

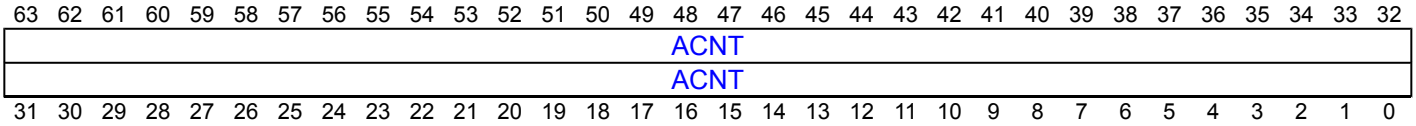
This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<n> are RES0.

Attributes

AMEVCNTR0<n> is a 64-bit register.

This register is part of the AMU block.

Field descriptions



ACNT, bits [63:0]

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 3.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0000000000000000000000000000000000000000000000000000000000000000'.

Accessing AMEVCNTR0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads of AMEVCNTR0<n> are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings:

When FEAT_AMU_EXT64 is implemented

[63:0] Accessible at offset `0x000 + (8 * n)` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

When FEAT_AMU_EXT32 is implemented

[63:0] Accessible at offset `0x000 + (8 * n)` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTR1<n>, Activity Monitors Event Counter Registers 1, n = 0 - 15

The AMEVCNTR1<n> characteristics are:

Purpose

Provides access to the auxiliary activity monitor event counters.

Configuration

External register AMEVCNTR1<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVCNTR1<n>_EL0[63:0].

External register AMEVCNTR1<n> bits [31:0] are architecturally mapped to AArch32 System register AMEVCNTR1<n>[31:0].

It is IMPLEMENTATION DEFINED whether AMEVCNTR1<n> is implemented in the Core power domain or in the Debug power domain.

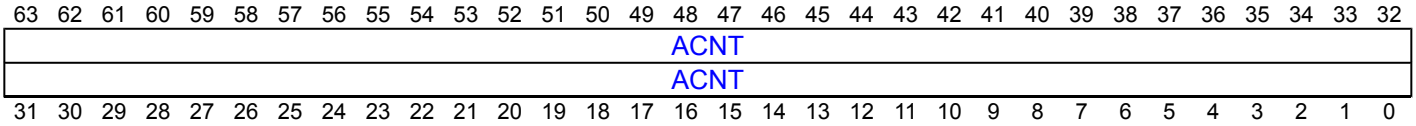
This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n> are RES0.

Attributes

AMEVCNTR1<n> is a 64-bit register.

This register is part of the AMU block.

Field descriptions



ACNT, bits [63:0]

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

The reset behavior of this field is:

- On an AMU reset, this field resets to '0000000000000000000000000000000000000000000000000000000000000000'.

Accessing AMEVCNTR1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads of AMEVCNTR1<n> are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

AMCGCR.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings:

When FEAT_AMU_EXT64 is implemented

[63:0] Accessible at offset `0x100 + (8 * n)` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

When FEAT_AMU_EXT32 is implemented

[63:0] Accessible at offset `0x100 + (8 * n)` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVTYPER0<n>, Activity Monitors Event Type Registers 0, n = 0 - 3

The AMEVTYPER0<n> characteristics are:

Purpose

Provides information on the events that an architected activity monitor event counter AMEVCNTR0<n> counts.

Configuration

External register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPER0<n>_EL0[31:0] when FEAT_AMU_EXT32 is implemented.

External register AMEVTYPER0<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVTYPER0<n>_EL0[63:0] when FEAT_AMU_EXT64 is implemented.

External register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER0<n>[31:0].

It is IMPLEMENTATION DEFINED whether AMEVTYPER0<n> is implemented in the Core power domain or in the Debug power domain.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n> are RES0.

Attributes

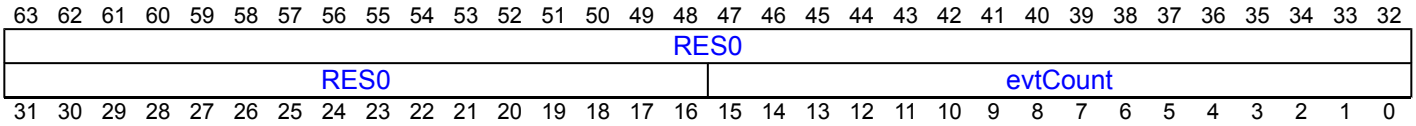
AMEVTYPER0<n> is a:

- 64-bit register when FEAT_AMU_EXT64 is implemented
- 32-bit register otherwise

This register is part of the AMU block.

Field descriptions

When FEAT_AMU_EXT64 is implemented:



Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

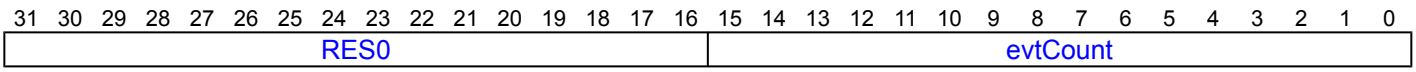
Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

evtCount	Meaning	Applies when
0x0011	Processor frequency cycles.	When n == 0
0x4004	Constant frequency cycles.	When n == 1
0x0008	Instructions retired.	When n == 2
0x4005	Memory stall cycles.	When n == 3

Access to this field is **RO**.

Otherwise:



Bits [31:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

evtCount	Meaning	Applies when
0x0011	Processor frequency cycles.	When n == 0
0x4004	Constant frequency cycles.	When n == 1
0x0008	Instructions retired.	When n == 2
0x4005	Memory stall cycles.	When n == 3

Access to this field is **RO**.

Accessing AMEVTYPER0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads of AMEVTYPER0<n> are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings:

When FEAT_AMU_EXT64 is implemented

Accessible at offset `0x400 + (8 * n)` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

When FEAT_AMU_EXT32 is implemented

Accessible at offset `0x400 + (4 * n)` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.

- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVTYPER1<n>, Activity Monitors Event Type Registers 1, n = 0 - 15

The AMEVTYPER1<n> characteristics are:

Purpose

Provides information on the events that an auxiliary activity monitor event counter AMEVCNTR1<n> counts.

Configuration

External register AMEVTYPER1<n> bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPER1<n>_EL0[31:0] when FEAT_AMU_EXT32 is implemented.

External register AMEVTYPER1<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVTYPER1<n>_EL0[63:0] when FEAT_AMU_EXT64 is implemented.

External register AMEVTYPER1<n> bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER1<n>[31:0].

It is IMPLEMENTATION DEFINED whether AMEVTYPER1<n> is implemented in the Core power domain or in the Debug power domain.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER1<n> are RES0.

Attributes

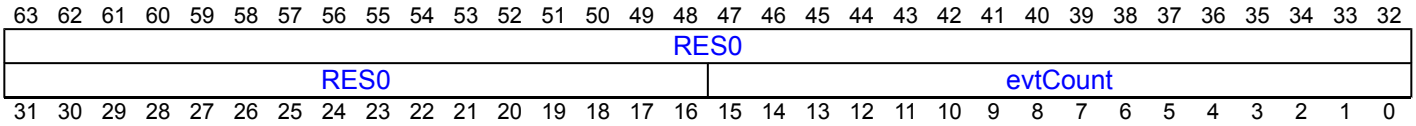
AMEVTYPER1<n> is a:

- 64-bit register when FEAT_AMU_EXT64 is implemented
- 32-bit register otherwise

This register is part of the AMU block.

Field descriptions

When FEAT_AMU_EXT64 is implemented:



Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

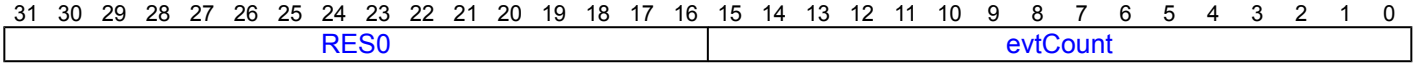
Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter AMEVCNTR1<n>.

It is IMPLEMENTATION DEFINED what values are supported by each counter.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:



Bits [31:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter AMEVCNTR1<n>.

It is IMPLEMENTATION DEFINED what values are supported by each counter.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVTYPER1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads of AMEVTYPER1<n> are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

AMCGCR.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings:

When FEAT_AMU_EXT32 is implemented

Accessible at offset `0x480 + (4 * n)` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

When FEAT_AMU_EXT64 is implemented

Accessible at offset `0x500 + (8 * n)` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMIIDR, Activity Monitors Implementation Identification Register

The AMIIDR characteristics are:

Purpose

Defines the implementer and revisions of the AMU.

Configuration

It is IMPLEMENTATION DEFINED whether AMIIDR is implemented in the Core power domain or in the Debug power domain.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMIIDR are RES0.

Attributes

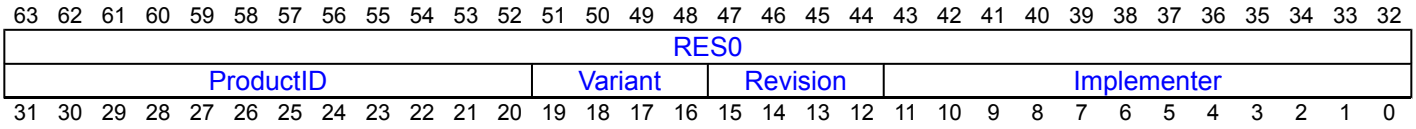
AMIIDR is a:

- 64-bit register when FEAT_AMU_EXT64 is implemented
- 32-bit register otherwise

This register is part of the AMU block.

Field descriptions

When FEAT_AMU_EXT64 is implemented:



Bits [63:32]

Reserved, RES0.

ProductID, bits [31:20]

This field is an AMU part identifier.

If AMPIDR0 is implemented, AMPIDR0.PART_0 matches bits [27:20] of this field.

If AMPIDR1 is implemented, AMPIDR1.PART_1 matches bits [31:28] of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [19:16]

This field distinguishes product variants or major revisions of the product.

If AMPIDR2 is implemented, AMPIDR2.REVISION matches AMIIDR.Variant.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

This field distinguishes minor revisions of the product.

If AMPIDR3 is implemented, AMPIDR3.REVAND matches AMIIDR.Revision.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the AMU.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for AMIIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

If AMPIDR4 is implemented, AMPIDR4.DES_2 matches bits [11:8] of this field.

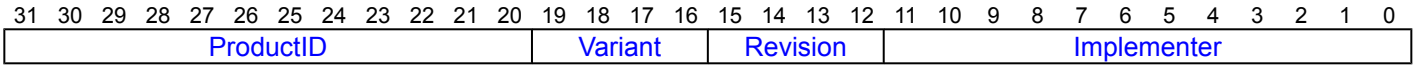
If AMPIDR2 is implemented, AMPIDR2.DES_1 matches bits [6:4] of this field.

If AMPIDR1 is implemented, AMPIDR1.DES_0 matches bits [3:0] of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:



ProductID, bits [31:20]

This field is an AMU part identifier.

If AMPIDR0 is implemented, AMPIDR0.PART_0 matches bits [27:20] of this field.

If AMPIDR1 is implemented, AMPIDR1.PART_1 matches bits [31:28] of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [19:16]

This field distinguishes product variants or major revisions of the product.

If AMPIDR2 is implemented, AMPIDR2.REVISION matches AMIIDR.Variant.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

This field distinguishes minor revisions of the product.

If AMPIDR3 is implemented, AMPIDR3.REVAND matches AMIIDR.Revision.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the AMU.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

Zero is not a valid JEP106 identification code, meaning a value of zero for AMIIDR indicates this register is not implemented.

For an implementation designed by Arm, this field reads as `0x43B`.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

If AMPIDR4 is implemented, AMPIDR4.DES_2 matches bits [11:8] of this field.

If AMPIDR2 is implemented, AMPIDR2.DES_1 matches bits [6:4] of this field.

If AMPIDR1 is implemented, AMPIDR1.DES_0 matches bits [3:0] of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMIIDR

Accesses to this register use the following encodings:

When FEAT_AMU_EXT64 is implemented

Accessible at offset `0xE08` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

When FEAT_AMU_EXT32 is implemented

Accessible at offset `0xE08` from AMU

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMPIDR0, Activity Monitors Peripheral Identification Register 0

The AMPIDR0 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

It is IMPLEMENTATION DEFINED whether AMPIDR0 is implemented in the Core power domain or in the Debug power domain.

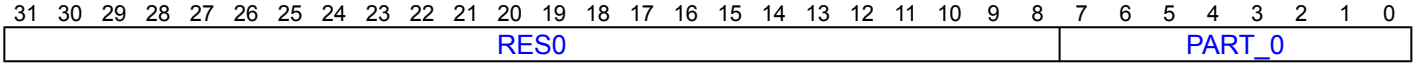
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMPIDR0. Otherwise, direct accesses to AMPIDR0 are RES0.

Attributes

AMPIDR0 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, least significant byte.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMPIDR0

Accesses to this register use the following encodings:

Accessible at offset `0xFE0` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMPIDR1, Activity Monitors Peripheral Identification Register 1

The AMPIDR1 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

It is IMPLEMENTATION DEFINED whether AMPIDR1 is implemented in the Core power domain or in the Debug power domain.

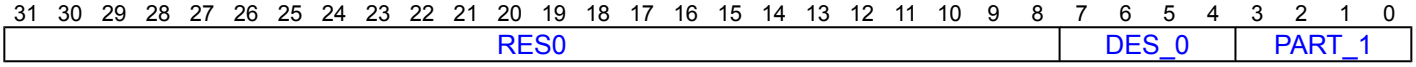
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMPIDR1. Otherwise, direct accesses to AMPIDR1 are RES0.

Attributes

AMPIDR1 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, least significant nibble of JEP106 ID code.

For Arm Limited, this field is `0b1011`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, most significant nibble.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMPIDR1

Accesses to this register use the following encodings:

Accessible at offset `0xFE4` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMPIDR2, Activity Monitors Peripheral Identification Register 2

The AMPIDR2 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

It is IMPLEMENTATION DEFINED whether AMPIDR2 is implemented in the Core power domain or in the Debug power domain.

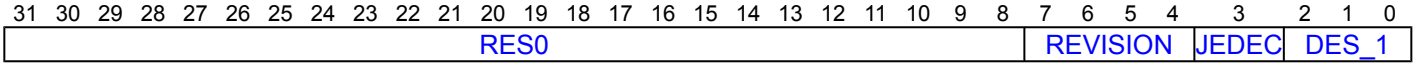
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMPIDR2. Otherwise, direct accesses to AMPIDR2 are RES0.

Attributes

AMPIDR2 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Part major revision. Parts can also use this field to extend Part number to 16-bits.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

Indicates a JEP106 identity code is used.

Reads as `0b1`.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, most significant bits of JEP106 ID code.

For Arm Limited, this field is `0b011`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMPIDR2

Accesses to this register use the following encodings:

Accessible at offset `0xFE8` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMPIDR3, Activity Monitors Peripheral Identification Register 3

The AMPIDR3 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

It is IMPLEMENTATION DEFINED whether AMPIDR3 is implemented in the Core power domain or in the Debug power domain.

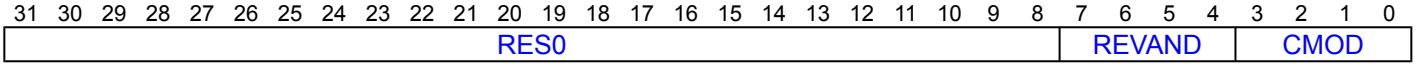
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMPIDR3. Otherwise, direct accesses to AMPIDR3 are RES0.

Attributes

AMPIDR3 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Part minor revision. Parts using AMPIDR2.REVISION as an extension to the Part number must use this field as a major revision number.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer modified. Indicates someone other than the Designer has modified the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMPIDR3

Accesses to this register use the following encodings:

Accessible at offset `0xFEC` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.

- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMPIDR4, Activity Monitors Peripheral Identification Register 4

The AMPIDR4 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

It is IMPLEMENTATION DEFINED whether AMPIDR4 is implemented in the Core power domain or in the Debug power domain.

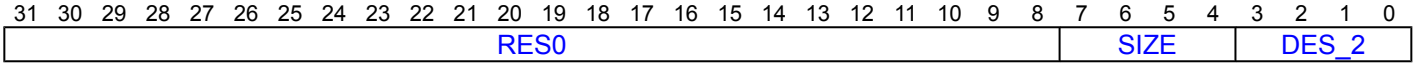
This register is present only when FEAT_AMUv1 is implemented and an implementation implements AMPIDR4. Otherwise, direct accesses to AMPIDR4 are RES0.

Attributes

AMPIDR4 is a 32-bit register.

This register is part of the AMU block.

Field descriptions



Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component. Log_2 of the number of 4KB pages from the start of the component to the end of the component ID registers.

Reads as `0b0000`.

Access to this field is **RO**.

DES_2, bits [3:0]

Designer. JEP106 continuation code, least significant nibble.

For Arm Limited, this field is `0b0100`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMPIDR4

Accesses to this register use the following encodings:

Accessible at offset `0xFD0` from AMU

- When boolean IMPLEMENTATION_DEFINED "AMU CoreSight management registers ignore access controls", accesses to this register are **RO**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Secure, and AMROOTCR.RA IN {0b001, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Realm, and AMROOTCR.RA IN {0b010, 0b000}, accesses to this register are **RAZ/WI**.
- When FEAT_RME is implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMROOTCR.RA != 0b011, accesses to this register are **RAZ/WI**.
- When FEAT_RME is not implemented, FEAT_AMU_EXTACR is implemented, an access is Non-secure, and AMSCR.NSRA == 0, accesses to this register are **RAZ/WI**.
- Otherwise, accesses to this register are **RO**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMROOTCR, Activity Monitors Root Control Register

The AMROOTCR characteristics are:

Purpose

Control register for Root, Realm, Secure, and Non-secure access to External AMU registers.

Configuration

It is IMPLEMENTATION DEFINED whether AMROOTCR is implemented in the Core power domain or in the Debug power domain.

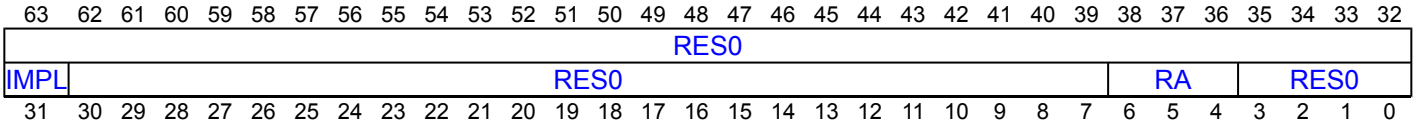
This register is present only when FEAT_AMU_EXTACR is implemented and FEAT_RME is implemented. Otherwise, direct accesses to AMROOTCR are RES0.

Attributes

AMROOTCR is a 64-bit register.

This register is part of the AMU block.

Field descriptions



Bits [63:32]

Reserved, RES0.

IMPL, bit [31]

IMPL	Meaning
0b1	Indicates AMROOTCR is present.

Access to this field is **RAO/WI**.

Bits [30:7]

Reserved, RES0.

RA, bits [6:4]

Register Access to all External Activity Monitors registers.

RA	Meaning
0b000	Root register access is enabled. Access from other address spaces is disabled, meaning accesses to all External AMU registers are RAZ/WI.
0b001	Root and Realm register access is enabled. Access from other address spaces is disabled, meaning accesses to all External AMU registers are RAZ/WI.
0b010	Root and Secure register access is enabled. Access from other address spaces is disabled, meaning accesses to all External AMU registers are RAZ/WI.
0b011	Root, Secure, Non-secure and Realm register access is enabled.

Other values are reserved.

For the CoreSight management registers, `0xFA8` to `0xFFC`, it is IMPLEMENTATION DEFINED whether these registers are RO or RAZ/WI when register access is disabled by this field.

Bits [3:0]

Reserved, RES0.

Accessing AMROOTCR

Accesses to this register use the following encodings:

Accessible at offset `0xE48` from AMU

- When an access is Root, accesses to this register are **RW**.
- Otherwise, accesses to this register are **RAZ/WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMSCR, Activity Monitors Secure Control Register

The AMSCR characteristics are:

Purpose

Control register for Secure, and Non-secure access to External AMU registers.

Configuration

It is IMPLEMENTATION DEFINED whether AMSCR is implemented in the Core power domain or in the Debug power domain.

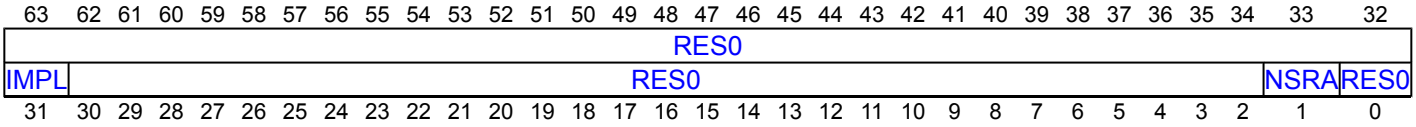
This register is present only when FEAT_AMU_EXTACR is implemented and FEAT_RME is not implemented. Otherwise, direct accesses to AMSCR are RES0.

Attributes

AMSCR is a 64-bit register.

This register is part of the AMU block.

Field descriptions



Bits [63:32]

Reserved, RES0.

IMPL, bit [31]

IMPL	Meaning
0b1	Indicates AMSCR is present.

Access to this field is **RAO/WI**.

Bits [30:2]

Reserved, RES0.

NSRA, bit [1]

Register Access to all External Activity Monitors registers.

NSRA	Meaning
0b0	Non-secure access is disabled, RAZ/WI.
0b1	Non-Secure access is enabled.

Bit [0]

Reserved, RES0.

Accessing AMSCR

Accesses to this register use the following encodings:

Accessible at offset `0xE40` from AMU

- When an access is Secure or an access is Root, accesses to this register are **RW**.
- Otherwise, accesses to this register are **RAZ/WI**.

2025-06-23 17:47:32, 2025-06_rel

Copyright © 2010-2025 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.